ISI/SR-83-23

University of Southern California

1982 ANNUAL TECHNICAL REPORT

July 1981 – June 1982

A Research Program in Computer Technology

INFORMATION SCIENCES INSTITUTE

213/822-1511

4676 Admiralty Way/Marina del Rey/California/90291-6695

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISI/SR-83-23	2. GOVT ACCESSION NO. AD-A127288	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 1982 Annual Technical Report: A Research Program in Computer Technology		5. TYPE OF REPORT & PERIOD COVERED Annual Technical Report July 1981 - June 1982 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) ISI Research Staff		8. CONTRACT OR GRANT NUMBER(s) MDA903 81 C 0335,
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE March 1983 13. NUMBER OF PAGES 138
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document is approved for public release and sale; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

.....

18. SUPPLEMENTARY NOTES

.....

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

1. implementation of interactive systems, knowledge base, knowledge-based inference, natural interface, online services, process script, service building, tool building, user interface
2. abstract formal specification, formal specification language, formal specification of systems, Gist, software development, specification mapping, symbolic evaluation, transformation implementation, VLSI design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes the research performed by USC/Information Sciences Institute from July 1, 1981, to June 30, 1982, for the Defense Advanced Research Projects Agency. The research applies computer science and technology to areas of high DoD/military impact.

The ISI program consists of thirteen research areas: *Cooperative Interactive Systems* - construction of a system to provide natural input/output and help facilities for users of interactive services; *Mapping Designs Between Language Levels* - develop transformations for converting high-level specifications of programs into implementations in software or VLSI; *Specification Acquisition From*

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. KEY WORDS (continued)

3. abstract programming, domain-independent interactive system, HEARSAY-III, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process information
4. design rules, device fabrication service, device testing, integrated-circuit oriented language, MOS Implementation System, silicon compilation, VLSI design, VLSI design library, wafer testing
5. computer mail, gateways, interconnection, internetwork protocol, multimedia mail, networks, protocol design, protocols, protocol verification, simple mail transfer protocol, transmission control protocol, type-of-service .
6. briefing aid application program, command graphics, computer graphics, high-level graphics language, network-based graphics, on-line map display
7. computer network, digital voice communication, network conferencing, packet satellite network, packet-switched networks, packet video, packet voice, secure voice transmission, signal processing, speech processing, vocoding
8. Ada, denotational semantics, formal definition, formal semantic definition, typed lambda calculus
9. AN/UYK-20, emulators, ISPS, microprogramming, MULTI, multimicroprocessor emulation, National Software Works, program development tools, QM-1, QPRIM, Smite
10. address space limitations, C, Interlisp, LISP dialects, UNIX, VAX, VMS
11. computing resources, network security, network survivability, strategic command, control and communication, user training
12. computer technology, human engineering, network-based processors, personal computers, user support
13. application software, ARPANET, customer service, hardware, interface, computer network, KA/KI, KL10/KL20, operations, PDP-11/45, resource allocation, system software, TENEX, timesharing, TOPS-20

20. ABSTRACT (continued)

Experts - study of acquiring and using program knowledge for making informal program specifications more precise; *VLSI--Services and Research* - development of a low-cost, fast turnaround LSI/VLSI device fabrication service to support VLSI research, and research on the VLSI design problem; *Internetwork Concepts* - exploring aspects of protocols for the interconnection of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; *Command Graphics* - development of a device-independent graphics system and graphics-oriented command and control applications programs; *Wideband Communication* - development of protocols and real-time systems to transmit digitized voice over the ARPANET and development of technology required for the future support of thousands of simultaneous conversations being transmitted over a wideband satellite channel in the internetwork environment; *Formal Semantics* - development of tools and methodologies to support the formulation of precise, readable, and accurate formal semantic definitions; *QPRIM* - production of an online interactive emulation facility housed in an existing mature operating system; *Interlisp* - development and maintenance of a portable, large address-space Interlisp implementation; *Support of a Strategic C3 System Experiment* - development of a detailed technical plan for the creation of a survivable message system and data bases through multiple copies of the critical components and data across the ARPANET and provision of the core facilities necessary to implement the plan; *New Computing Environment* - investigation and adaptation of developing computer technologies to serve the research and military user communities; and *Computer Research Support* - operation of TENEX and TOPS-20 service and continuing development of advanced support equipment.

ISI/SR-83-23

University of Southern California

1982

ANNUAL TECHNICAL REPORT

July 1981 - June 1982

A Research Program in Computer Technology

Principal Investigator and Executive Director:
Keith W. Uncapher

Deputy Director:
Thomas O. Ellis

Prepared for the Defense Advanced Research Projects Agency

Effective date of contract 1 July 1981
Contract expiration date 30 June 1984
Contract # MDA 903 81 C 0335
ARPA order 4242

This research is supported by the Defense Advanced Research Projects Agency under Contract No. MDA 903 81 C 0335 ARPA Order No. 4242

Views and conclusions contained in this report are the authors and should not be interpreted as representing the official opinion or policy of DARPA, U.S. Government or any other person or agency connected with them.

This document is approved for public release and sale; distribution is unlimited.

INFORMATION SCIENCES INSTITUTE

213/822-1511

4676 Admiralty Way/Marina del Rey/California/90291-6695

RESEARCH AND ADMINISTRATIVE SUPPORT

Institute Administration:
Robert Blechen
Kathleen Fry
Maureen Jester
Julie Kcomt
Gina Maschmeier
Rolanda Shelby
Steve Wagner

Graphics Design:
Curtis Nishiyama

Librarian:
Sally Hambridge

Publications:
Jim Melancon
Sheila Coyazo

Secretaries to Directors:
Joyce K. Reynolds
Suzanne Robb

CONTENTS

SUMMARY

This report summarizes the research performed by USC/Information Sciences Institute from July 1, 1981, to June 30, 1982, for the Defense Advanced Research Projects Agency. The research applies computer science and technology to areas of high DoD/military impact.

The ISI program consists of thirteen research areas: *Cooperative Interactive Systems* - construction of a system to provide natural input/output and help facilities for users of interactive services; *Mapping Designs Between Language Levels* - develop transformations for converting high-level specifications of programs into implementations in software or VLSI; *Specification Acquisition From Experts* - study of acquiring and using program knowledge for making informal program specifications more precise; *VLSI--Services and Research* - development of a low-cost, fast turnaround LSI/VLSI device fabrication service to support VLSI research, and research on the VLSI design problem; *Internetwork Concepts* - exploring aspects of protocols for the interconnection of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; *Command Graphics* - development of a device-independent graphics system and graphics-oriented command and control applications programs; *Wideband Communication* - development of protocols and real-time systems to transmit digitized voice over the ARPANET and development of technology required for the future support of thousands of simultaneous conversations being transmitted over a wideband satellite channel in the internetwork environment; *Formal Semantics* - development of tools and methodologies to support the formulation of precise, readable, and accurate formal semantic definitions; *QPRIM* - production of an online interactive emulation facility housed in an existing mature operating system; *Interlisp* - development and maintenance of a portable, large address-space Interlisp implementation; *Support of a Strategic C3 System Experiment* - development of a detailed technical plan for the creation of a survivable message system and data bases through multiple copies of the critical components and data across the ARPANET and provision of the core facilities necessary to implement the plan; *New Computing Environment* - investigation and adaptation of developing computer technologies to serve the research and military user communities; and *Computer Research Support* - operation of TENEX and TOPS-20 service and continuing development of advanced support equipment.

EXECUTIVE OVERVIEW

USC Information Science Institute (ISI) is a large, university-based information processing research center. ISI's focus is on basic science, applied research, and systems, both in information processing and related computer-based communication. In addition, the Institute remains committed to establishing new levels of achievement in the support and delivery of shared and personal computing environments, often tailored to specific needs. One major project during the past year has been the design of new computing support environments for ISI research staff and for the external user community. ISI remains especially active and productive in technology transfer to military departments. Further, ISI has been unusually successful in uncovering military opportunities and problems, which in some instances have formed the basis for new research initiatives.

In addition, as a major supplier of high-quality computing resources over the ARPANET, ISI serves both the military and the university research communities. User training, user support, and protocol development augment the delivery of the basic computer resources.

The DARPA-sponsored research programs at ISI are as follows: Cooperative Interactive Systems, Mapping Designs Between Language Levels, Specification Acquisition From Experts, VLSI--Services and Research, Internetwork Concepts Research, Command Graphics, Wideband Communication, Formal Semantics, QPRIM, Interlisp, Support of a Strategic C3 System Experiment, New Computing Environment, and Computer Research Support. These projects form four major research directions: *software production technology*, *user-friendly systems*, *computer-based communicatons*, and *Very Large Scale Integration*. The diversity of research interests at ISI provides a broad base for a healthy interchange and amplification of ideas and information.

Cooperative Interactive Systems. The Consul system is designed to provide a natural, consistent interface for the user services of an interactive environment. User activities such as sending messages, maintaining an appointment calendar, generating graphical displays, now handled by separate subsystems, will all be accessible through a single interface that allows natural language input and provides help to the user. This is achieved through knowledge-based inference on a detailed model of user and system behavior. Research issues include knowledge representation and inference techniques, acquisition of domain-dependent knowledge, explanation, and software methodology. The current prototype system demonstrates natural interaction with a message service.

Mapping Designs Between Language Levels. Formal specifications cannot become an integrated part of the software development process until they cease being merely documentation. They must become the seed from which the algorithm, expressed in some programming language, is developed. The atomic constructs of the specification language must be re-expressed in terms of the more primitive constructs of the programming language. Identifying the distinctions between these languages and providing mappings under user guidance is one major thrust of the effort. The second major thrust of this effort is to apply this same mappping technology to a different design task, namely VLSI design. The needs here are just the same as in software design--to have multiple layers of specification which adequately express certain aspects of the design but leave others to be handled by other layers, and to provide a means of using one layer to develop another. However, because the field is so new, the appropriate specification layers are much less well known than with software. Hence, In addition to developing mappings between the layers, we are also defining the formal language to be used for each layer.

Specification Acquisition From Experts. This work has been directed at helping people create unambiguous, consistent, and complete formal program specifications through informal description. An early version of the developed system successfully converted several small, informal specifications into formal specifications. When the time came to attempt larger specifications, some fundamental changes were required in the knowledge base and in the compiler paradigm. These changes brought about a refocusing of effort, which resulted in the creation of a new type of formal specification language based on formalized versions of commonly occurring natural language constructs and the start-up of two new research projects to capitalize on the groundwork laid by SAFE.

VLSI--Services and Research. This project is aimed at the advancement of a low-cost, fast turnaround VLSI device fabrication service to support the geographically distributed DARPA and NSF supported VLSI research communities, which have no direct access to VLSI fabrication, but have access to computer communication networks. A special effort has been made to implement a VLSI fabrication service called MOSIS (MOS Implementation System). The motivation for this service is to provide an "almost interactive" VLSI chip design environment with a design-to-packaged VLSI device turnaround time of a few weeks. We believe this service is causing a revolution in system architecture research as dramatic as the introduction of interactive computation was to programming. The current major thrust of the MOSIS operation is the support of advanced CMOS with small feature size.

Internetwork Concepts Research. This project explores the design and analysis of computer-to-computer communication protocols in multinetwork systems. The project has three task areas: (1) Analysis, (2) Applications, and (3) Design and Concepts. Protocol Analysis is concerned with the correctness of protocols, in particular Transmission Control Protocol (TCP). Protocol Applications is concerned with the development of demonstration internetwork applications, in particular a prototype computer message system. Protocol Design and Concepts is concerned with the development of network and transport protocols, in particular the Internet Protocol and TCP, and seeks new approaches in the application of packet switching to communication problems.

Command Graphics. As more command and control information is maintained in computer form, computers will need to take a more active role in the presentation of that information. To facilitate the decision-making process, online computer-generated color graphics will replace binders of batch-generated printer listings. The purpose of this project has been to develop a system architecture to meet current and future C2 graphic requirements, with particular attention paid to adaptability to available computation, communication, and display resources, usability in a transnetwork environment, and ability to support the creation of pictures for use outside the immediate application environment. The work has resulted in the definition and implementation of a C2 Graphics System on the ARPANET. Work this year has included the transfer of the graphics system to small, powerful 16-bit microprocessors and the development of a briefing-aid application.

Wideband Communication. The ISI Network Secure Communication (NSC) Project has been instrumental in the development of protocols and real-time systems to transmit digitized voice over the ARPANET, both in point-to-point conversations and multisite conferences. The project is now broadening its scope as the Wideband Communication (WBC) Project, which will develop the technology required for the future support of thousands of simultaneous conversations being transmitted over a wideband satellite channel in the internetwork environment. It will advance packet voice from a demonstration program to an experimental system continuously available for use in the transaction of normal daily business. While the NSC project concentrated on voice communication, the WBC Project will work on integrated communication of several media, including voice. The goal is

to develop real-time multimedia teleconferencing using wideband packet-switched networks. The initial emphasis other than voice will be on the development of a video bandwidth compression system which operates in real time and takes advantage of the ability of a packet-switched network to accommodate varying bandwidth requirements.

Formal Semantics. The principal goals of this project are the development of tools and methodologies for supporting the development of precise, readable, and accurate formal semantic definitions of programming languages. The specific research focus of the project is building tools for manipulating, processing, implementing, and testing the formal definition of Ada written by a group at the Institut National de Recherche en Informatique et en Automatique in France.

QM-1 Programming Research Instrument. The QPRIM effort is producing an online interactive emulation facility housed in an existing mature operating system. QPRIM aims to bring program execution and testing into the programmer's working environment without paying the typically prohibitive cost involved in utilizing a simulation program on the development host. The PRIM project built such a prototype facility within the TENEX operating system; that facility was operational at ISI from 1974 until 1979. QPRIM is a PRIM-like emulation facility running under the DEC TOPS-20 operating system and utilizing a production-emulation engine, the Nanodata QM-1. An interface unit between the DECsystem 20 and the QM-1 was installed in March 1979. The facility has been running and available at ISIB since June 1980. As part of the system shakedown, an emulator for the AN/UYK-20 will be written and tested.

Interlisp. This project has developed and is maintaining a portable, large address-space Interlisp implementation. The first version was done for the DEC VAX computer.

Support of a Strategic C3 System Experiment. DARPA has defined an experiment in strategic C3 systems to be conducted in cooperation with the WWMCCS System Engineer and the Strategic Air Command (SAC). The concept of the experiment is to demonstrate and evaluate the use of new technologies (such as the ARPANET, packet radio, network security, and distributed knowledge-base techniques) for strategic command, control, and communication. The DARPA experiment is defined as having three phases: Phase 1 is planned to demonstrate air-to-surface packet radio links and gateways into the ARPANET as a first step in evaluating the feasibility of a truly survivable strategic network. Phase 2 is directed toward creating a survivable message system and data bases through multiple copies of the critical components and data across the ARPANET. Phase 3 will address the feasibility of rapid reconstitution of a strategic network by deployment of packet radio networks to reconnect surviving elements of the network. ISI is assisting in the development of a detailed technical plan for Phase 2 and providing the core facilities necessary to implement the plan.

New Computing Environment. The goal of this project is to adapt developing computer technologies to serve the research and military user communities. The resulting model computing environment will serve four purposes: (1) It will provide a very significant improvement in languages, system support, and additional computing cycles to the ongoing ISI research effort. (2) It will serve as a testbed and eventual existence proof for the implemented technology. (3) It will serve as a proving ground for local computer environments targeted for ARPA and the military community, as well as a device for trying out new research ideas that will eventually be adapted by those communities. (4) The small size, portability, and local computing capability of PCs will allow for experimentation and realization of Command and Control requirements for an environment that can exist in mobile command centers, be they vans, ships, or airplane based.

Computer Research Support. ISI supports, operates, and maintains one TENEX and five TOPS-20 systems at ISI on a schedule of 164 hours per week. TENEX/TOPS-20 service is provided both to ARPA and to its research projects via the facilities at ISI. ISI also operates one TENEX and one TOPS-20 system at a computer center that is part of the Command and Control Testbed at the Naval Ocean Systems Center, San Diego, Calif. The Institute provides support for the Penguin at ARPA-IPTO, NLS user support, and NLS software support. The Institute also provides remote training and documentation for its military users in order to allow them to make the most effective use of the available facilities.

1. COOPERATIVE INTERACTIVE SYSTEMS

Research Staff:
William Mark
Thomas Lipkis
William Swartout
David Wilczynski

Research Assistants:
Stephanie Forrest
Gabriel Robins

Support Staff:
Lori Holzer

Cooperative Interactive Systems is an effort to build an experimental system for allowing natural interaction between users and computer services. The **Consul** system is a knowledge-based user interface environment that includes facilities for understanding natural language requests and producing natural language help and explanations. Providing this kind of natural interaction requires the system to have detailed models of users and services--and the ability to use inferential knowledge to map back and forth between these models. And, because these models are large and detailed, Consul must have the ability to build them "semi-automatically," i.e., to acquire models on the basis of information solicited from experts. Consul research therefore focuses on representation, inference, and acquisition for knowledge in the interactive systems domain.

1.1 PROBLEM BEING SOLVED

User interface technology has not kept pace with the growing demand for interactive computer services. Interfaces to interactive services still do not follow general conventions, offer adequate help and documentation, or provide sufficient flexibility--much less accept the user's natural input form, explain errors, or answer user questions. In current interactive systems, individual services provide their own interface conventions and command language. There are few examples of services that offer useful help as an integrated part of normal operation. Command languages, function keys, and menus are deficient in their ability to express the variations and combinations of users' needs. User requests may therefore go unanswered--even though the desired functionality is present in the actual service--because of a mismatch between the user's expression and the system's expectation.

Writing user interface software to solve these problems is an enormous task--too great a burden for the individual service builder in almost all cases. The interface is inevitably a large, complex program (often half the code in the service). Furthermore, interactive services almost always require adaptation to changing user needs. Interface software must prevent perturbations in the underlying service environment from causing changes in the style and semantics of user interaction.

In summary, though the transfer of computer services to "real user" environments is now technically and economically feasible, the transfer will fail unless the service interface is easy and natural to use.

1.2 GOALS AND APPROACH

The objective of the Consul system is to provide a natural interface for its users across a wide range of functional services. Its natural interface includes natural language request understanding and explanation facilities for users of interactive services such as electronic mail, automated appointment

calendar, and document preparation. The system is intended to be used by a wide class of users ranging from novice to experienced.

Consul includes a methodology for building services into its natural interface facility. Users' needs change constantly, and vary greatly from one environment to another. The needs of any particular group of users must be satisfied by service builders familiar with those needs. However, the task of building an interactive service that provides a natural interface is too great to be left as a burden for individual service builders. Consul offers an acquisition mechanism for automatically incorporating new service features (or changes to existing features) into its natural interface facility through system-directed dialogue with the service builder.

The Consul approach is based on representation of the knowledge that allows a natural interface: models of the characteristics and behavior of users and services. The resulting knowledge base of user and service actions, objects, and events provides a foundation for the inferential activities that are necessary to map the user's description of his needs into the system's descriptions of service capabilities and requirements (and back to the user's domain again to produce explanation responses). Using the knowledge base to provide interface facilities is a process of inferring the appropriate system behavior for a given input. Consul's basic operating mode is as follows: the user types a request into the system; the request is parsed, i.e., rendered into the system's knowledge representation; inference is used to see the representation of the request as a description of some system action; this action, a service operation invocation or explanation response, is then "executed," fulfilling the user's request. Consul's other operating mode is acquisition. Here the input is a service builder's description of a function or data structure, and inference is used to see it in relation to the known descriptions in Consul's user and system models. All of this behavior is the result of the operation of the fundamental inferential processes of *classification*, *realization*, and *mapping* on the knowledge base (see Figure 1-1 and [6]).

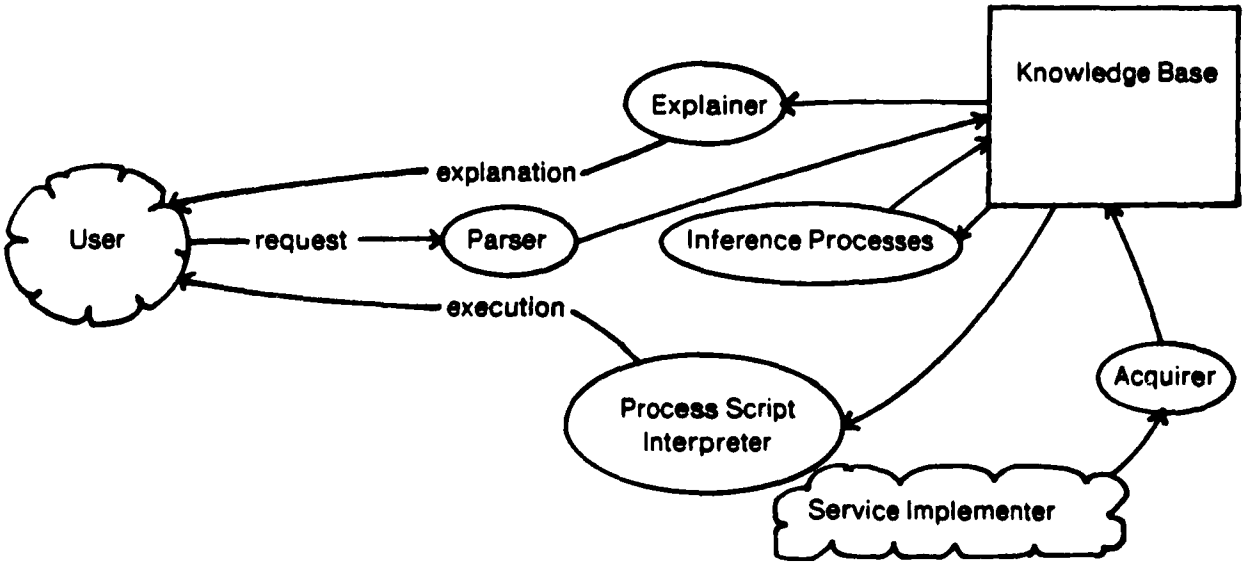


Figure 1-1: The Consul system

1.2.1 Modelling

Consul's knowledge base is implemented in the KL-ONE representation formalism [2]. The knowledge base contains models of the objects, actions, and events pertinent to users, interactive systems in general, and specific interactive services.

The user model provides a consistent description of a "user view" of the system. User objects, actions, and events may have simple or rather complex relationships with objects, actions, and events in the systems model. For example, a *user message* corresponds directly with a *system message*, but is a much simpler description. On the other hand, a *user deletion event* is a quite different description from a series of system events (*mark*, *collect*, *expunge*) it could be said to correspond to. Since psychological and human factors research has not yet provided enough experimental data about computer users, Consul cannot portray the known viewpoint of a particular user community or communities (though this is certainly our intent in the long-term). Instead, we have developed a more-or-less general user model--and made sure that none of our reasoning techniques rely on its precise features. We will be ready to update our model as research results become available.

The systems model is a representation of the detailed behavior of the basic operations found in any service (e.g., deletion, scheduling, display), along with the data structures these operations work on (files, tables, display lists, etc.). This general knowledge of interactive systems must be particularized to the actual operations and data structures of each service that is implemented in Consul. For example, the model for a particular mail service must describe *specific* (actually executable) functions like "DeleteMesssage" and "ShowMessage," and specific objects like "MessageFile" and "MessageDisplayHeader"). This service knowledge is built into Consul *semi-automatically*, i.e., as a result of the system directed acquisition dialogue with the implementers of that service [8].

The actual programs that implement the functionality of the service are expressed in terms of **process scripts** [5]. Process script programs consist of two parts: a procedure to perform some action and some declarative information about that procedure. The declarative part is in the form of a small number of categories of information required by the acquisition mechanism in order to see how the function represented by the process script specifies its arguments and effects its results (see Figure 1-2). Dialogue with the service builder is carried out in terms of these descriptive elements.

```
ProcessScript  SIGMAForward;
  Input                      u:SIGMAUser;
  Output                     none;
  DataStructuresAccessed     SIGMAOpenMessage,SIGMALoggedOnUser;
  Preconditions              SIGMAOpenMessageSV = true;
  SideEffects                none;
  Undo                       none;
  Error Conditions           e:NoMailBoxForRecipient;
    Call SIGMASend(u, "Forwarded");
```

Figure 1-2: A process script

Since it would be tedious for the service builder (and overwhelming for Consul) to extend the acquisition dialogue down to the most detailed level of implementation, Consul gives the programmer a mechanism for describing some aspects of his service as black boxes as far as Consul is concerned. The connection between these black boxes and the process scripts of the service is in terms of **process atoms**--identical to process scripts except that they contain, instead of a

procedure body, a call to a piece of application code written in any language supported by the host machine (Figure 1-2 actually shows a process atom).

1.2.2 Inference

All Consul behavior is based on the fundamental inference activities of recognition and redescription. Recognition is the process of determining how each piece of incoming information fits in with the system's current knowledge. Redescription is the process of determining how each recognized piece of knowledge can be viewed as applicable to the system's problem-solving goals.

In Consul, recognition is divided into **classification**, finding the appropriate place for each new piece of information in Consul's taxonomy of knowledge, and **realization**, finding the real world objects described by each new piece of information. Classification involves the determination of the relationship between the terms of a new description and the terms already known to Consul. Thus, a user request to forward a message must be broken down into its component terms ("a request whose object is sending a previously received message to a set of recipients") and related to known terms ("request," "sending," "message," etc.). When classification is complete, the request is seen to be subsumed by a known class of user requests, and therefore subject to further processing.

Realization is the association of each new term with the real world objects it describes. In the message forwarding request, the term "message" must be associated with whatever actual message the user wishes to forward. This is done on the basis of existing descriptions of real world objects. If the user has asked to forward "the message I just received from Smith," the realizer would look for the real world object described as a message, as most recently received by the user, and as sent by Smith. Classification and realization are intimately related; their joint processing implements recognition in the Consul system.

Consul's redescription or **mapping** process also consists of two related processes: rule application and alternative description. Rules in Consul are represented in the knowledge base: they are simply linkages between two ordinary knowledge base structures, one acting as the "condition," the other as the "conclusion." Interpretation of the rule is straightforward: an entity whose description instantiates the condition can also be described as an instantiation of the conclusion (with relevant pieces of the inital description "mapped" across according to the specified linkages in the rule). The process of rule application thus depends on classification: when a description is classified in Consul's knowledge base, it may be seen to instantiate the condition of a rule. The rule can then be used to generate a new description, which is in turn classified, perhaps instantiating another rule condition, and so on until a description is created that corresponds to one of Consul's external behaviors: answering a request (via process script execution or explanation) or acquiring a new piece of knowledge.

If no rules apply to the current description of the entity of interest, and if the description does not correspond to an external behavior specification, mapping can proceed by finding an alternative description of the entity. Multiple descriptions of entities, indeed any descriptions of entities, are due to the workings of the realization process during recognition. The redescription mechanism is simply taking advantage of this prior realization activity in its attempt to find a description of the entity that is amenable to one of Consul's external behavior processes. The alternative description is thus used exactly as the initial one, to be subject to rule application and eventual external action.

1.2.3 Handling Natural Requests

Consul's major activity is understanding and answering user requests. These requests can be in the form of natural language, requiring parsing (including "semantic analysis") before any Consul inferential activity can begin. That is, the parser (developed at Bolt Beranek and Newman, Inc. [1]) must translate the user's natural request form into the system's knowledge base form.

For example, the parser's interpretation of the request

Forward this message to Jones.

would be constructed as a specialization of Consul's knowledge base representation of "requests to send things." Understanding of the parsed user request is achieved through application of the basic inference mechanisms. Each input to the system is first classified in the knowledge base according to its relationship with the knowledge that is already there. Any possible realizations for the new description are made at that time. The initial classification produced by the parser is rarely sufficient to allow the system to take action (i.e., there is no "forward this message to Jones" operation). The user's request must therefore be redescribed (and reclassified) if the system is to know how to handle it. In this case, redescription proceeds via the following steps:

- since the initial classification of the request does not show it to be amenable to either execution or explanation, Consul looks for rules that apply to it;
- in this case, the only rule that applies is a very general one that says "a request for some user action can also be described as the invocation of the system operation whose objective is the same as the objective of the user action, and whose effect is the same as the effect of the user action";
- this results in a redescription of the original request as an invocation of the system operation whose objective is "forwarding" and whose effect is that "this message" will be "at" "Jones";
- the new description is classified; again it can be neither executed nor explained--but this time, no rule applies (because the system does not know what to do with a description of the form "this message");
- Consul therefore looks for alternative descriptions based on prior realizations; in this case, realization has found that the description "this message" refers to a message the user has just drafted;[1]
- trying this substitution, a rule is found to apply: "operations whose objective is a kind of 'moving' and whose effect puts a known system object 'at' a known system location can also be described as 'move operations'" (this rule did not apply previously because "this message" was not a known system object);
- rule application results in a new description as the invocation of a move operation of the "forwarding" type whose object is "the message that resulted from the most recent message composition event" and whose destination is "Jones";
- classification of this description determines that it is not amenable to execution; this is due to the fact that the message to be forwarded is a draft message (the result of the composition event), and no mail system function will forward a draft message (messages must be received in order to be forwarded); the description is also not amenable to explanation; however, a rule applies;
- the rule states that "invocations of operations whose type and parameters are known to the system should produce an explanation response if they cannot be executed"; the logic here is that the user must have asked for something reasonable (since Consul has mapped it into something it knows about), but for some reason the operation cannot be executed--Consul must determine the reason and explain it to the user;

---

[1] This realization corresponds to "deictic resolution" in the usual natural language processing sense.

- the result of this rule application is a description that is amenable to explanation: request processing is complete.

The actual formulation of the explanation response is described in the next section. The need for explanation is caused by the fact that "this message" happened to refer to a draft. If it had referred to a received message, the move operation description generated by the second rule application would have been executable, the description would have been passed to the process script interpreter, and the user's request would have been accomplished.

1.2.4 Producing Explanations

Producing explanations in response to user requests is an inferential activity in its own right. As shown above, the request understanding process is responsible for determining whether or not a request requires an explanation response (either because the user asked for one directly or because the system was unable to execute his request). After it has been determined that some form of explanation is necessary, Consul must formulate the appropriate response.

In the case of direct requests for assistance ("What has to be in a message?", "Why did my file disappear?", etc.), response formulation is a matter of finding the answer in system terms and then redescribing the answer in user terms. This is necessary because the user's question refers to some aspect of system requirements or behavior: the answer can come only from an analysis of the features in question. However, the user will not necessarily be able to understand an answer in that framework. It must be redescribed in *his* framework in order to be understandable. For example, the answer to "Why did my file disappear?" should not be "The archive server just ran, and the file was outside of the kept version range," but something more like "In order to save space, old versions of files are periodically placed in backup storage. Your file can be retrieved at your request."

Requests that receive explanation responses because they cannot be executed must be treated differently. Here the user's intent is the performance of some action; Consul's problem is to determine what the user intended to do (*target finding*), why the intended action could not be performed (*contradiction*), and whether there are actions that could be performed to satisfy the user's intent (*suggestion*). It must then explain this all to the user.

For example, in the case of "Forward this message to Jones," response formulation requires:

- *Target Finding*: Based on the initial parse of the request, Consul finds that the user was trying to perform that user action whose objective is "sending." Consul therefore collects all executable system operations whose objective is also "sending"; there are four.
- *Contradiction*: Consul compares the final redescription of the request (i.e., the move operation of the "forwarding" type) with each of the targets, and finds that it would have classified as a system forwarding operation if the message had been received rather than been a draft.
- *Suggestion*: Since the user has asked to perform the operation on a particular message, Consul will not suggest that he change this part of his request (i.e., it will not say "You can't forward that message to Jones, but you can forward this other message to him"). It therefore sees if any other of the "sending" type operations could work on his original message; two of them can.

The response has now been formulated. A natural language generation program uses it to produce

the response "You cannot forward a draft message. You can 'release' it or 'send it for review' [the two relevant sending operations]." The generator makes sure (by examining the user model) that the user is expected to know about "release" and "send for review" before making this response. If the user were not expected to recognize them by name, a more descriptive response would have been generated.

1.2.5 Acquiring Services

Ultimate interpretation of user requests, in particular the ability to execute or explain requests that refer to individual service capabilities, requires detailed models of services. It would be virtually impossible for us to build in all of these models by hand. It would also be impossible for the service programmer to build them by hand, since he is presumably unfamiliar with our model and, indeed, the entire modelling process. It is therefore essential that Consul provide a computer-aided methodology for incorporating detailed models of services into the overall system.

Consul integrates new services into its environment via an acquisition dialogue with the service builder [8]. Implementing services in the Consul environment includes describing them in terms of Consul's model of interactive systems. The acquisition dialogue is a step-by-step determination of the relationship between Consul's general model of a given operation and its implementation in terms of the process script the service builder is proposing. The process script programming formalism has been especially designed to provide an adequate function description for allowing the dialogue mechanism to ask the service builder appropriate questions about his program.

For example, the acquisition mechanism compares the process script in Figure 1-2 with the system's model of send operations. The dialogue will be in terms of the information supplied in the process script descriptors: e.g., acquisition will ask what corresponds to its model of the "sender" of the send operation; the service builder will answer that it is the *SIGMALoggedOnUser* specification in **DataStructuresAccessed**. This process continues until every part of Consul's model of send has been related to the process script.

It will often be the case that an aspect of Consul's model will not correspond directly with any part of the service builder's process script. For example, in Consul's model of the send operation, messages end up in the mailbox of the recipient. In the process script of Figure 1-2, the destination of the message is not explicitly represented: it is buried in the (black box) workings of the code itself. Acquisition must uncover the appropriate relationship between the model's required destination and the information supplied in the process script. It must then represent this relationship as an inferential link between some elements of the process script and the necessary aspects of the model. This "inferential link" is in fact a rule to be used by Consul's mapping process. In this example, the acquisition mechanism constructs a rule expressing the relationship between users and their mailboxes (taking advantage of the data structure model to note that any user who can receive messages must "own" a mailbox). This rule will allow later requests to send mail to "users" to be correctly translated into requests to send mail to the relevant mailboxes. Acquisition therefore builds not only descriptive models of service functions, but also an important subset of the rules required to handle natural requests and explanations.

1.3 TECHNICAL ISSUES

Consul processing relies on the presence of a detailed representation of the elements of Consul's user and system domains and the inferential links between them. This translates into a need for a very large, highly structured knowledge base. A major issue for Consul is finding a workable technology for the construction, maintenance, and use of this large knowledge base.

Construction of the knowledge base means modelling beliefs, events, actions, and objects. This task can be divided into three parts: deciding which concepts to include, deciding what to express about them, and deciding how to implement that expression.

- Deciding what to model is a strategic design decision: "more" modelling increases the system's capability, but is costly in terms of execution speed and machine requirements; "less" modelling provides a clean break between what the system does and does not know, but brings up issues of how the system handles what it does not know. We are trying to expand our use of the "less" modelling approach, since we feel it has the best potential for long-term payoff.
- Deciding what to express about a concept is now and will probably always remain an art: it is essentially the problem of describing something so that someone else (or in this case, something else) can understand it. We are gaining experience with this process, and have made considerable progress, especially through the use of abstract datatypes to consistently describe the objects of the system world.
- Deciding how to represent the knowledge is a scientific/philosophical problem of developing a consistent formalism. We, in association with our colleagues at Fairchild Research, Xerox Palo Alto Research Center, and Bolt Beranek and Newman, Inc., have greatly improved the clarity and efficiency of our formalism through careful preservation of the epistemological structures inherent in the knowledge we are trying to represent.

Maintenance of the knowledge base is largely a problem of providing the appropriate tools for creating, examining, and changing knowledge structures. We have produced useful tools for each of these activities, but still have much to do in order to make our maintenance procedures smooth and efficient.

Use of the knowledge base requires the discovery and implementation of efficient inference and search procedures. We need algorithms for rapidly determining the relationships among terms in the formalism, using terms to make statements about the world, and using inferential meta-knowledge to pursue the problem-solving goals of our system. The classification, realization, and redescription mechanisms probably represent the state of the art in this area. Nonetheless, they are only a first-cut solution of the overall problem. We are currently investigating new algorithms based on our latest thinking about the representation formalism and on the probable future availability of parallel computation methods.

Finally, Consul is investigating a set of technical issues not directly related to the knowledge base: methods for constructing the underlying service software. This is the software engineering problem of building programs and data structures with sufficient flexibility and consistency to meet the demands of responding to arbitrary user requests. We have found the abstract datatype formalism to be a useful aid in implementing data, and are actively exploring mechanisms for enforcing consistent programming practice to meet Consul's overall service design requirements.

1.4 SCIENTIFIC PROGRESS

- o We have implemented a new Consul system on the Xerox Dolphin that represents a significant expansion and improvement on previous versions. Nonetheless, it is still very much a prototype which works only on a very limited number of test cases. The prototype consists of:
  - a network knowledge base of over 800 concepts in the KL-ONE knowledge representation formalism;
  - the latest RUS/PSI-KLONE parser from BBN, adapted for use in Consul;
  - an enhanced inference facility, including a more thorough classification algorithm, a new realization component, and an improved rule application mechanism; this facility also represents a 100-fold improvement in execution time over the initial algorithms, despite its wider scope and enhanced functionality;
  - an acquisition component, including a stylized dialogue capability;
  - an explanation facility for responding to user requests that the system cannot perform, including a (rather basic) natural language generation capability that produces English sentences from KL-ONE structures.
- o We have produced major new facilities for the KL-ONE formalism in general as well as specific enhancements to Consul's use of that formalism for representation and inference. We have also participated in significant basic knowledge representation research with groups at Fairchild Research, Xerox Palo Alto Research Center, and Bolt Beranek and Newman Inc.
- o Our publications and presentations include the following:
  - 1981 International Joint Conference on Artificial Intelligence (two papers) [6, 8]
  - 1982 SHARE Conference [9]
  - 1982 National Computer Conference [7]
  - Hewlett-Packard Research Colloquium
  - 1981 KL-ONE Workshop (five papers) [3]

In addition, we have built a set of support tools for constructing and updating KL-ONE network knowledge structures. Our work in this area has included the construction of a remote file server for accessing knowledge structures and other information stored on remote machines.

We have investigated the use of parallel control structures for our knowledge base algorithms [4] in order to take advantage of the kind of VLSI-based parallel hardware that we believe will become available in the near future.

1.5 IMPACT

Consul provides a new, complete approach for building interactive systems with good user interfaces. User interfaces based on natural communication open up computer systems to a much wider audience--eventually to anyone with a need for their services. The natural interface accepts requests from users with absolutely no previous training with the system. It can explain how to use the system, what the system did, or what has gone wrong in terms that the user can understand and learn from. This kind of interaction is virtually a necessity for acceptance by users in many aspects of society where the need for computer service is greatest. Natural user interface technology is fundamental to continued progress in the delivery of computer services to society at large.

In the shorter term, the Consul-style interface can greatly increase the usability of computer systems for current target user communities. Natural language processing allows the user expressiveness not found in command language systems. Consul's parser and inference mechanisms are designed to handle both full English sentences and more succinct, stylized forms of expression. Consul's inference facility allows it to adapt to different user input characterizations depending on the user's experience with a service. This is possible because of the system's model of the underlying semantics of *all* requests, based on its knowledge of user needs and system functional capabilities. That is, it is possible to infer the user's intent from a wide range of input forms expressing that intent. Also, Consul's knowledge base serves as a repository from which to provide help, explanation, and documentation about a service's static characteristics and dynamic behavior.

Change in interactive systems is inevitable. Modification of existing services (bug-fixing, new features) and the introduction of new services are constant, expected activities in the interactive computing world. Consul provides an environment in which change can take place with minimum disruption of the system's interface with the user. Consul's process script formalism maintains a separation between rapidly changing functional elements and the stable knowledge base that drives the user interface. Consul's acquisition mechanism ensures that changes to services are accounted for in terms of that knowledge base. As a service builder defines a process script for a service, Consul tries to classify it in the existing system knowledge base. Once classified, the script inherits all of Consul's knowledge about the kind of operation it specializes. New functions added in this way will fit a user's model of interactive services.

The Consul system is a prototype for the kind of user interface that will be demanded by the users of future interactive systems. Consul's acquisition methodology allows service builders to take advantage of this advanced user interface facility as they develop and change their services to meet future needs.

1.6 FUTURE WORK

Our future efforts will be concentrated on increasing both the breadth and depth of Consul's user interface and service acquisition capabilties.

A major goal is the completion of a Consul system that is robust enough to provide a useful response for *any* user input. Where possible, the system will provide the service execution or explanation response that the user has requested. Where this is not possible, the response generated will always be a useful suggestion for how the user can accomplish his intent. In the short-term, natural language processing capabilities will be limited (i.e., there will be many natural language inputs that the system cannot fully understand). We expect natural language processing power to increase dramatically as we expand our models and input handling techniques.

In addition, we are planning a major push in the area of explanation, with the goal of enhancing the current prototype to include more sophisticated explanation strategies and the ability to handle question-and-answer style help.

In parallel with this natural language work, we will explore the use of window/menu and limited-syntax command input as an integral part of the Consul interface. This combined approach will take advantage of the new personal workstation technology to provide an interface that reacts very efficiently to the most common forms of input, while remaining flexible and robust for natural

language, help, and error input. The resulting system will be a very practical, yet very powerful, interface for a wide class of computer users.

All work in expanding Consul capabilities, especially in the area of robustness, relies on more complete models of users and systems. We are therefore deeply involved in the modelling process, expecting to eventually increase our knowledge base size by an order of magnitude, to the 10,000 concept level.

This increase in model size and sophistication in turn depends on substantial improvements in our underlying knowledge representation formalism and the tools we have developed to use it. We, in cooperation with others in the KL-ONE community, are therefore engaged in basic representation research in order to enhance the capability and efficiency of our underlying formalism. We are also actively pursuing more development-oriented tool building efforts to make it easier to build and maintain our models.

Finally, we are concentrating on increasing the scope and speed of our basic inference mechanisms (classification, realization, and redescription) as well as the activities that use them (request handling, explanation, and acquisition). This involves the application of our techniques to new areas, as well the refinement of their use in current applications.

REFERENCES

1. Bobrow, R., and B. Webber, "Knowledge representation for syntactic/semantic processing," in *Proceedings of the National Conference on Artificial Intelligence*, AAAI, August 1980.

2. Brachman, R., *A Structural Paradigm for Representing Knowledge*, Bolt Beranek and Newman Inc., Technical Report, 1978.

3. Schmolze, J., and R. Brachman (eds.), *Proceedings of the 1981 KL-ONE Workshop*, Fairchild, Technical Report 618, May 1982.

4. Forrest, S., "Consul Note 15: A Parallel Algorithm for Classification in KL-ONE Networks," USC/Information Sciences Institute, August 1982.

5. Lingard, R., "A software methodology for building interactive tools," in *Proceedings of the Fifth International Conference on Software Engineering*, 1981.

6. Mark, W., "Representation and inference in the Consul System," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, IJCAI, 1981.

7. Mark, W., "Natural-language help in the Consul system," in H. L. Morgan (ed.), *AFIPS Conference Proceedings, Volume 51: National Computer Conference*, pp. 475-479, AFIPS Press, June 1982.

8. Wilczynski, D., "Knowledge acquisition in the Consul system," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, IJCAI, 1981.

9. Wilczynski, D., "Building an advanced, general-purpose user interface: Progress on the Consul system," in *SHARE Proceedings*, pp. 5-11, March 1982.

2. MAPPING DESIGNS BETWEEN LANGUAGE LEVELS

Research Staff:
Robert Balzer
Martin Feather
Jack Mostow
David Wile

Research Assistant:
Monica Lam

Support Staff:
Audree Beal

2.1 PROBLEM BEING SOLVED

For several years under ARPA sponsorship and through related projects, our group has been developing a foundation for a new software development paradigm based on an increased formalization of the software life cycle and increased involvement of computer tools as aids and checks for people engaged in this development enterprise.

We have developed and are beginning to exploit an advanced formal system specification language (Gist) which allows complex systems of multiple interacting participants to be defined abstractly in enough detail that they can be simulated without overly constraining the possible realizations of the subsystem to be implemented.

This simulation capability enables the specification itself to act as a prototype of the desired subsystem and its environment (the rest of the specified system) so that its logical behavior (as opposed to its performance characteristics) can be investigated prior to implementation to insure that it matches the user's informal intent. Towards this end, we are developing both an interpreter of the formal specification language which will enable the specification to be tested much as programs are currently tested and a symbolic evaluator (patterned after the "meta-Evaluation" facility which forms the semantic base of the SAFE project) which will enable the logical behavior of the desired subsystem on selected classes of inputs to be simultaneously ascertained.

We are also developing an interactive transformation system which will allow an implementor to map a formal specification into an efficient implementation through a structured development of refinement steps, each based on a decision made by the implementor. The system will automate the program manipulations to carry out the implementor's decision, to prepare the program for the chosen decision, to simplify the resulting program, and to document the decision as an aid to later maintenance.

When all these capabilities are developed, a new software development paradigm will be almost possible. It will be based on the formal specification of systems, computerized validation tools to enable these formal specifications to act as prototypes to ensure they match the user's intent, and an interactive design process to map these specifications in a series of stages into an efficient implementation and to automatically record that design for later maintenance.

The key to this approach, and its major departure from current practice, is a suitable abstract formal specification. Since such specifications represent a considerable amount of effort, they will

only be created if this effort is minimized (one of the SAFE project's goals) and if they can actually be useful in the development of systems.

Unfortunately, one critical component of this approach is missing: the actual transformations which map a formal specification into an implementation. Our efforts so far have focused on developing the framework needed to support this approach. This project is developing a catalog of transformations required to make this framework useful. It is addressing two separate domains, software development and VLSI design.

2.2 GOALS AND APPROACH

This project will identify recurrent conceptual language-independent optimization issues, catalog broad methods for dealing with these issues, develop formal transformations to implement these methods, and provide guidance for users in selecting among alternative design decisions.

A major goal is to structure the system and the mappings so that users achieve with this approach as efficient a program as they could previously. (The framework effort has focused on improving the reliability of the implementation and the ease of obtaining it.) Since the user will be interactively selecting the mappings to apply there is no requirement that the system be able to make the sections itself, only that a comprehensive set of mappings exist or, as an escape, that the user can bypass the mapping mechanism to make necessary modifications. Such escapes will be documented as unverified manual mappings.

2.2.1 Advanced Specification Mappings

This project will identify the conceptual distinctions between advanced formal specification languages and existing programming languages to determine those recurring implementation and optimization issues which these existing languages force programmers to decide before expressing the algorithm in their language.

These issues include the following:

1.General purpose constraints
how and where to efficiently check them, how to strengthen the algorithm so that they cannot be violated.

2. General purpose inferencing
when should the computation be performed, should it be saved (this is the basic store/recompute issue), how long is it valid (what invalidates it).

3. Representation selection
which data items should be aggregated into larger units, which access paths between items should be made inexpensive.

4. Search optimization
(i.e., finding an object, or all objects, that satisfy some predicate) how can properties of the object(s) desired guide the search, can or should heuristic techniques be employed.

5. Historical data what data about past states of the system is needed, what auxiliary structures should be employed to maintain it.

We will codify existing techniques for dealing with these issues into alternative methods and develop formal transformations to implement them.

This will provide the methodology for mechanically mapping, under user guidance, from a specification language into an existing programming language. The use of formal specifications as a standard part of software development is critically dependent upon such a methodology because the specifications would then become the actual starting point of the development process, the seed from which the implementation is developed, rather than merely a piece of documentation disconnected from the development itself.

This mapping from a specification to some existing programming language will actually be done in two stages. The first will be the steps we have described previously: the sequential application of user selected transformations which re-express the freedoms afforded by the specification language in terms of other more primitive constructs. The critical issue here is that this re-expression is in the same language. "Higher-level" constructs are being replaced by lower level ones. Since both are part of the same language, they can be intermixed freely. This allows the mapping to proceed gradually, piece by piece. Once all the "high-level" constructs have been re-expressed, the resulting program will be mechanically block translated into an existing programming language.

A second reason for splitting this mapping into two stages is that the re-expression mappings are largely target language independent. While there are some small differences among existing target programming languages in terms of which subset of the specification language they support (e.g., tasks), the similarities at the semantic level are much greater than the differences. We also feel that the differences in relative cost of the same construct in different target programming languages is a second order effect. Thus, by keeping the user-guided portion of the mapping in terms of a subset of the same specification language, the system is not only simplified by not mixing two different languages, but these techniques and the mappings themselves also become applicable to multiple target languages.

We believe that this general approach to design, in which the freedoms afforded by a formal specification language are incrementally re-expressed in terms of an implementable subset of that language, which is then mechanically block translated into an implementation language, is much more widely applicable than just software development. In particular, this idea will be explored in terms of VLSI design as part of the second mapping effort within this project as described in the "VLSI Design" section below.

2.2.2 VLSI Design

As critical as design issues are in software, they pale in comparison with those in VLSI. The experience with this technology is miniscule compared with the relatively modest experience with software. The number of experts in the field is tiny. The paradigms and idioms of the technology are just being discovered. Tool support is primitive. Yet the chip technology is proceeding at an even faster pace than computer hardware in general. There is a growing realization that the utility of this field will be design, rather than hardware, limited.

The cost of designing chips must fall by about three orders of magnitude if million gate chips are to become a reality for other than memory and general purpose processors. While the gain in software productivity doesn't come close to this goal, the complexity of the underlying physics is so great that sealing this level off is a critical first step. This isolation of the underlying physics via the use of "design rules," such as those proposed by Mead and Conway, and the isolation of the physical layout problem, such as in the Bristle Blocks and the Silicon Compiler efforts, represent the main focus (and rightly so) of the VLSI community.

However, it is our contention that, as critical as this activity is, it is merely the foundation upon which a methodology for dealing with the many levels of design needed to connect conceptual design to the level at which automatic implementation can take over. Furthermore, we contend that this multi-level design activity is highly analogous to the software design activity we described above, and that the same TYPE of system is needed for both. The validity of this similarity critically depends upon the assumption that the physical layout and underlying physics issues have been sealed off through the creation of one or more appropriate design languages. We take this goal of the VLSI community as our starting point and consider the issues of multi-level designs which terminate at this level.

Given this assumption, the main distinction between VLSI design and software design appears to be primarily the centrality of time and concurrent processing in VLSI. Our work on specification languages (Gist) has already identified the importance of these issues for system specifications, so we feel that much of our approach and background will be appropriate for these higher levels of VLSI design. It should be noted that at these conceptual levels, discrete circuit design is also very similar (except for scale and the relative cost of active components versus communication). Thus, this more established domain may well offer indications of appropriate design levels and idioms.

Our main tasks then will be to identify appropriate levels of design specification, to develop or adapt formal languages suitable for these levels, and to develop mappings between these levels. As with the advanced specification mapping effort within this project, these mappings will focus on making implementation decisions which remove the freedoms afforded by one level which are not suppported at lower levels.

2.3 ACCOMPLISHMENTS

2.3.1 Advanced Specification Mapping

We have developed alternative mappings (i.e., different ways of realizing higher level constructs in terms of lower level mechanisms) for several of the specifications freedoms provided by Gist. These include

2.3.1.1 Historical reference

Historical reference refers to the ability to extract information from any preceding state in the computation history. The ability to do this frees the specification from determining in advance (and remembering) all current information that might be required at some time in the future. Required information is merely retrieved at the point of consumption without concern for when it is available.

Mapping away historical references

Options for mapping historical references away are:

- introducing and maintaining auxiliary data structures to hold information which might be referenced, and modifying the historical references to extract the information from these introduced structures. The requirement for economy of storage in an implementation encourages the implementor to seek a compact representation for the information that need be preserved, and to discard information once it is no longer useful.
- resolving the historical reference by derivation in terms of information available in the current state (without having to retain extra information from past states).

The latter is rarely an available option. When it is, the two alternatives above present the classical space/time tradeoffs; we must still compare the cost of the derivation with the cost of storage and maintenance of redundant information to permit simple access.

Two factors bear directly on the range of choice of how much, and what, information needs to be retained in an implementation of historical reference. These are the nature of the retrievals of the information, and how it is used. In general we must compare the frequency of storing historical information with the frequency of accessing it to determine whether it is better to perform only required calculations on such historical functions upon storage or upon retrieval

2.3.1.2 Constraints and non-determinism

Constraints within Gist provide a means of stating integrity conditions that must always remain satisfied. Within Gist, constraints are more than merely redundant checks that the specification always generates valid behaviors; constraints serve to *rule out* those behaviors that would be invalid. In conjunction with the use of non-determinism they serve as an extremely powerful specification mechanism, permitting us to describe an activity in a non-deterministic fashion; those behaviors of the activity leading to states that violate the constraint are "pruned away." This provides the ability to express our intents more directly (in the form of constraints), rather than encoding all the processes of the specification so as to interact in only those ways that prevent arriving at an undesirable state.

Mapping away constraints and non-determinism

To fully appreciate the implementation freedoms afforded by the specification constructs of constraint and non-determinism, it is necessary to understand the distinction between non-determinism in the specification and non-determinism in the implementation. The first permits alternative behaviors, while the second permits alternative ways of producing a particular behavior.

The most interesting case for consideration in deriving an implementation arises when specification non-determinism and constraints combine to describe deterministic behavior (as in the case of determining the correct way to set a switch). A range of possibilities present themselves for mapping away such non-determinism and constraints.

- The "predictive" solution: At one extreme we might seek to introduce code into all the non-deterministic choice points to perform the necessary calculations. These determine the choices that will not lead to disaster arbitrarily far into the future.
- The "backtracking" solution: At the other extreme we might derive a backtracking algorithm, with the choice points as the backtracking points and the constraints mapped into search-terminating checks at the appropriate places in the program.

The choice of method is affected very strongly by the nature of the domain of the specification. The capabilities of the effectors (if there are any in the system being specified), the amount of information available for making decisions, and the desired amount of precomputation all affect the choice of algorithm. We see a general preference for avoiding backtracking-type implementations if at all possible, and producing instead a "predictive" solution.

The issue concerning the methods by which an implementation preserves the integrity of constraints is one of the more difficult mapping problems. At the specification level, constraints apply to the whole specification. As implementors, however, we only have control over the behavior of the portion of the specification we are implementing. We assume that our mission is to implement this portion in such a way as to insure that such constraints are not violated independent of the behavior of other portions of the system as long as they too meet their specifications. This means, for example, that we cannot expect the environment to behave benevolently. However, *some* requirements are imposed on the environment and these are invariants we rely on rather than have to maintain. But it is our responsibility to implement our portion to react to any allowable behavior of the other portions of the specification so as to never allow any constraints to be violated. Of course, such a one-sided division of responsibility may not be possible in a given situation (i.e., the specification is unimplementable). Avoiding constraint violation may require cooperation among the portions of the system. If so, then the specification must be revised so that such cooperation is required from each portion and can therefore be relied upon.

2.3.1.3 Derived relations

One of the underlying features of Gist is that all knowledge is represented in terms of objects and relations among those objects. Change is therefore represented by the creation/destruction of objects, and the insertion/deletion of relations among them.

Often it is convenient to make use of a relationship that is derived from others. Its derivation is declared once and for all, and serves to denote all the maintenance necessary to preserve the invariant between the derived relation and the relations upon which it depends.

Mapping away derived relations

The specificational power of this construct comes from being able to state a derivation in a single place, and make use of the derived information throughout the specification. Since no corresponding construct is likely to be available in any implementation language we might choose, we must map the derivation into explicit data structures and mechanisms to support all the uses of that information scattered throughout the program. We have a wide range of choices in how we might do this mapping.

- At one extreme we might simply unfold the derivation at all the places where a reference to the relation is made. Having done this, we may completely discard the relation and its derivation. This approach is analogous to backward inference, where computation is performed on demand and at the site of the need.
- At the other extreme we might retain the relation, but scatter throughout the program the code necessary to explicitly maintain the invariant between the derived information and the information upon which it depends ("base" information). Thus, it is necessary to find all possible places where a change is made to any of the base information, and augment such places with code to recalculate the derived information. This approach is analogous to forward inference, where computation is performed whenever a modification to a relevant predicate occurs and at the site of the change.

As with mapping away historical reference, the nature of the use of derived information affects our options and decisions.

The choices among the implementation alternatives imply alternative trade-offs between storage and computation in the resulting program. Completely unfolding the derivation is tending towards complex recalculation with a minimum of stored data. Maintaining relations is tending towards simplifying calculations by simple maintenance of additional data. Maintaining just the required portions of a relation is a compromise between these positions.

In performing the recalculation when base information changes we may be able to take advantage of the state of knowledge prior to the change to incrementally update the derived information. For example, simply appending a new element onto the end of a sequence is easier than recalculating the entire sequence afresh. This is an example of a general technique we call "incremental maintenance," and is derived from the work of other researchers in set-theoretic settings, particularly Paige and Schwartz [4] and Paige and Koenig [5], who call the technique "formal differentiation," and Earley [1], who calls it "iterator inversion."

In some places the introduced code can readily be eliminated, since although the base information is changing, the derived relation can be determined to always remain unchanged.

Sometimes, it is possible to relax the restriction that the invariant embodied by the relation definition need hold at all times. When maintaining a derived relation, if the uses of that relation do not immediately follow the changes to the base information, we may delay the incremental maintenance, provided that it is performed before any retrieval of the information. Thus, the invariant only need hold at the times it is "used." To achieve this effect we may either apply sophisticated special-purpose transformations that deal with such delayed computation for maintenance, or apply more standard ones that do "normal" incremental maintenance and then apply code moving transformations to relocate the maintenance operations. The need for this may arise if incremental maintenance leads to inserting code into other portions of the specification--those over which we have no control.

2.3.1.4 Demons

Demons are Gist's mechanism for providing data-directed invocation of processes. A demon's trigger is a predicate, which triggers the demon's response whenever a state change induces a change in the value of the trigger predicate from false to true.

Demons are a convenient specification construct for use in situations in which we wish to trigger an activity upon some particular change of state in the modeled environment. They save us from the need to identify the individual portions of the specification where actions might cause such a change and the need to insert into such places the additional code necessary to invoke the response accordingly. The specificational power of the demon construct is enhanced by the power of Gist's other features, since the triggering predicate may make use of defined relationships, historical reference, etc.

Mapping away demons

Demons triggered by some external event--that is, the relevant state changes are produced only in the environment portions of the specification--are a special case. The only mapping option is in how to implement the low-level details of detecting that event--for example by using a polling loop, or an interrupt-driven implementation. More interesting from the mapping point of view are demons triggered by the portion of the specification to be implemented. The implementation must explicitly connect the production of the conditions upon which to trigger these demons with the invocation of the response.

Mapping away such a demon involves identifying all places in the program where a state change might cause a change of the value of the demon's triggering predicate from false to true, and inserting the code to make such a determination and perform the demon's response when necessary. This mapping has much in common with that for maintaining derived relations. Here, however, the information to be maintained is a truthvalue; its definition is the trigger predicate of the demon. Whereas in a derived relation a change would cause a maintenance operation of the information, here the demon's response is to be invoked.

Consideration of mapping away a demon which is randomly triggered dramatically illustrates the effect that the order in which constructs are mapped away may have on the development. If that demon is mapped away when its trigger is still random (i.e., it could trigger at any time), the result would be to augment *every* state change with a non-deterministic choice of whether or not to invoke that demon's response. Further manipulation of the resulting specification would be severely hampered, particularly in attempting the removal of constraints by limiting non-determinism. Obviously it is far better to map away the non-determinism while it is localized in the demon (as discussed in constraints and non-determinism above), and then to map away the demon.

2.3.1.5 Total information

For specificational purposes it is convenient to assume free use of any information about the world described by the specification. When implementing a specification, however, not all of that information is necessarily available at all times. (For example, it may be distant and take time or expense to move it to where it is consumed; it may be obtainable only through a hardware device that has limited availability or applicability.)

Mapping away reliance on total information

Mapping away this reliance is similar to mapping away historical reference. It can be done either by introducing and maintaining auxiliary data structures to hold information that we might need to reference, or by finding ways to derive the required information from other information that is available.

In some cases the lack of perfect knowledge may constrain our choice of implementation, by forcing us to select only those implementations guaranteed to make no reliance upon information that cannot be derived from the observable behavior.

As with mapping away other constructs, the context in which unobservable information is used, and the nature of its use may affect our decisions on how to do the mapping. Quite often there are potential interactions between how we do this and other kinds of mappings.

Complete example

These mappings were used to mechanically translate a postal package-routing system specified in Gist into a compilable form. This formal development is presented in a paper "Implementing Specification Freedoms" by London and Feather to appear in *Science of Computer Programming*.

2.3.2 VLSI Design Mappings

In the VLSI portion of the Mapping project, we have decided to concentrate on the portion of the design that converts an algorithm into a functional-level circuit.

We identified three classes of decisions made in this phase of the design:

1. Operations used in the algorithm must be allocated to components and times. Operator allocation is complicated by the fact that an operation may be distributed in time and space. For example, a multiplication inside a loop may be implemented by re-using a single multiplier, or by replicating the multiplier and executing the loop body simultaneously for all values of the loop index in parallel.
2. The data flow between components must be made explicit. An algorithm can refer to the value of a variable simply by naming it, but in VLSI accessing a variable requires wiring a connection to the point of use from wherever the variable is stored.
3. Control mechanisms must be designed to implement high-level control constructs used in the algorithm, such as iteration and complex conditionals, in terms of hardware constructs, such as registers and combinational logic.

2.3.2.1 Design Language Requirements

Now consider the requirements for a design language in which to represent these decisions transformationally. It must be able to

- represent the initial algorithm.
- represent the intermediate steps in the design process. In particular, the language should make it easy to write transformation rules to account for these steps.
- represent the result of the design process, a functional-level circuit consisting of a collection of hardware cells and their interconnections.

To simplify the language problem, we have restricted ourselves to clocked (synchronous) circuits for now. In particular, we assume a two-phase clocking scheme that reads registers and computes combinational functions in phase 1, and writes registers in phase 2.

2.3.2.2 Modelling Hardware Design as Program Transformation

We adopt an approach in which circuits are represented as programs. Not all programs represent circuits, however, only programs that satisfy certain restrictions corresponding to facts of life in the VLSI domain. The design process that maps an algorithm to a circuit is thus represented as applying a series of program transformations to an ordinary program to get it into the restricted form that corresponds to a circuit.

The mapping from this restricted class of programs to functional-level circuits is defined as follows:

1. Procedures represent hardware modules.
2. Procedure arguments represent module input and output ports.
3. "Own" variables (local variables that retain their value between successive calls to the procedure containing them) represent registers.
4. Procedure declaration nesting represents module nesting; subroutines correspond to submodules. (Thus recursive procedures are prohibited, as one might expect--a circuit cannot contain itself.)
5. Calling a procedure with a list of actual input arguments corresponds to placing the values of those arguments at the input ports of the module and invoking the module. The symbol any is used to denote don't-care values.
6. Returning a list of values corresponds to placing them at the output ports.
7. Assignment statements denote register transfers.
8. References to variables correspond to reading registers or input ports.

2.3.2.3 A Simple Example: Accumulate a Sum

To illustrate how circuits can be represented as programs, consider the problem of summing a sequence of values x_1, ..., x_n and putting the result in some variable; call it Total. This can be done by a simple program:

```
Procedure Accumulate(S);
        local Sum;
        begin
        Sum ← 0;
        For each x in S do Sum ← Sum + x;
        Return Sum;
        end;
end Accumulate;

Total ← Accumulate(<x1, ..., xn>)
```

This program can be interpreted as an abstract description of a hardware module that sums its inputs, but it fails to make explicit such details as how the module implements the sequential control of the begin-end block and the For statement, and how the set S is input. One hardware implementation of Accumulate is depicted in Figure 2-0, and is represented by the following program:

```
Procedure Accumulator(ctrl, x);
        Own Sum;
        Case ctrl of
                reset:  Sum ← 0;
                add:    Sum ← Sum + x;
                no-op:  ;
                report: return Sum;
                end case;
end Accumulator;

begin
Accumulator(reset, any);
For each x in <x1, ..., xn> do Accumulator(add, x);
Total ← Accumulator(report, any);
end
```

The body of the procedure describes what the Accumulator module does in a single clock cycle,

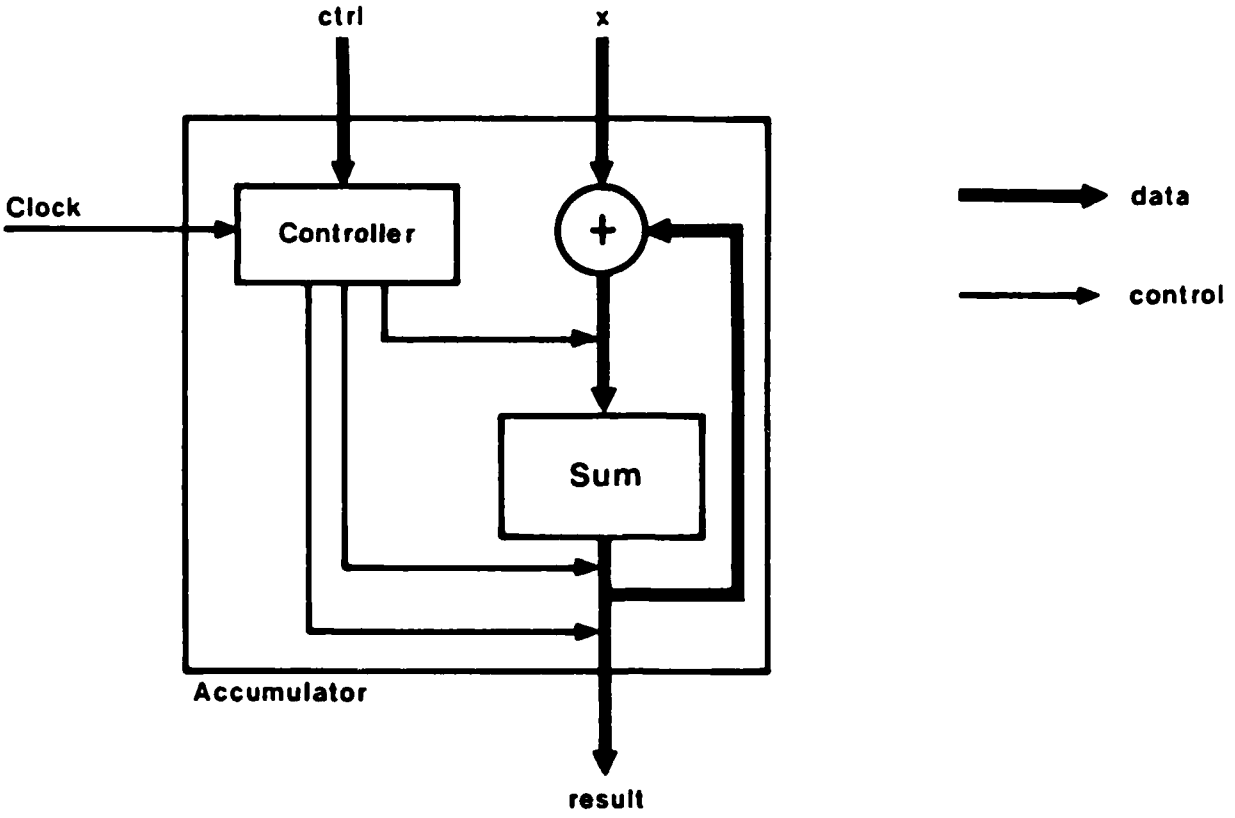


Figure 2-1: A hardware implementation of Accumulator

and the driver code for calling it describes the calling sequence required to make the module compute the desired result, i.e., what inputs to give it in successive cycles and when to get the result from the output.

The module's basic action is to input the value x and add it to the running sum stored in the register Sum (case "add"). However, the module must also be able to reset Sum to zero (case "reset"), to leave it unchanged in cycles when no value is presented (case "no-op"), and to report its value on demand (case "report"). Thus the module has an output port, called TotSum, and a second input, called ctrl, that tells the module which action to perform. This is represented in the procedure by a case statement that selects which action to perform based on the value of the ctrl input. In addition, the module has a controller that decodes the ctrl input and regulates the initialization and updating of the Sum register and the output of the result. This mechanism is not explicitly modelled in the procedure, which represents the module at a level of abstraction in which switches and bus structure are left implicit.

Observe that the problematic control constructs in the first program (begin-end blocks and iteration) have been extracted into the driver in the second one. Unlike the Accumulate procedure, the Accumulator procedure has a straightforward hardware interpretation. What properties does a program need in order to have such an interpretation? This question is the subject of the next section.

2.3.2.4 What Subclass of Programs Represent Circuits?

In order to correspond to a circuit, a program must satisfy several restrictions imposed by hardware constraints:

- *No data collision*: Parallelism is allowed, but simultaneous assignments to the same variable are prohibited, as they must be in any parallel algorithm if its behavior is to be deterministically defined.
- *Explicit allocation*: Concurrent calls to the same procedure are generally prohibited, since the same module cannot accept two different sets of inputs simultaneously. In such cases, the module must be replicated, with one copy for each caller. This is represented by an array of procedures--an odd construct for a programming language, but a reasonable one for representing hardware.

  This restriction is actually overly severe; multiple callers can call the same module simultaneously provided they don't use the same data lines. Thus an exception to the prohibition on concurrent calls is made in the case of arbitrator-like modules that are called simultaneously by several callers but on different input lines. A special case is an output-only module such as a time-of-day server. Another exception is pipelining: a multi-stage module can process several overlapping calls, provided none of them tries to use the same data line at the same time.
- *Explicit data flow*: No global variables are allowed, and a procedure body may refer only to its input arguments and local **own** variables. This is because in VLSI, accessing a variable requires a physically wired path to the place where it is stored. To model these paths, we require that non-local values be explicitly passed as arguments.
- *Explicit control*: A procedure body must be a finite state machine that executes one transition per clock cycle and stores all its state in its local variables; all functions computed in a single cycle must be combinational. Functions that satisfy this restriction have a straightforward hardware analog. More complex functions must be implemented by breaking them down into a series of single-cycle computations.

We have developed a collection of transformations for mapping ordinary programs into programs that satisfy these restrictions, and have implemented several of them in Interlisp, making use of its Pattern Match Compiler [6]. The transformations can be classified according to the type of decisions they make: allocation, data flow, or control. Some example transformations are presented below.

2.3.2.5 Allocation Transformations

Suppose we wish to implement in hardware a For loop in an algorithm:

```
For each x in S
        do <body>;
```

One way is to build a module M that inputs x and performs the loop body, and then invoke this module for each x in sequence:

```
Procedure M(x); <body>; end M;

For each x in S in sequence
        do M(x);
```

This allocation scheme is called *sequential re-use*, as indicated by the keyword **sequence**.

An alternative is to build one module for each element x_1, ..., x_n of S, and invoke all the modules in parallel:

```
Procedure M[i in 1..n](x); <body>; end M;

For i from 1 to n in parallel
        do M[i](x_i);
```

This allocation scheme is called *parallel replication*. It requires that executing <body> for one element x does not affect the result of executing it for another element. Note that in the expression $M[i](x_i)$, the index i distinguishes among modules in the array, while the argument x_i represents a value passed to an input port in module M[i]. One might choose to eliminate this port by hardwiring the value x_i into the module.

A third alternative, pipelining, is discussed in [3], but further work is required to fit it into the program representation used here.

2.3.2.6 Data Flow Transformations

The transformation program we have implemented contains one interactive transformation, called MakeArgsExplicit, for making data flow explicit. Suppose we have decided to implement in hardware an algorithm containing a nested function call f(g(x)), where the variable x is defined external to the function call. Some module F will compute the expression f(g(x)). The data flow question (in its simplest form) is, will module F input x and compute g(x) itself, or will the value of g(x) be computed externally and supplied as input? In general, the functions may take multiple arguments and the function calls in the algorithm may be more deeply nested, but this is the basic idea. The MakeArgsExplicit transformation systematically traverses an expression being implemented in a module, asking the user whether each sub-expression that uses data external to the module will be computed inside or outside the module. Based on this information, it determines the input ports for the module.

Such decisions may appear unimportant: after all, g(x) must be computed somewhere, so what does it matter how the computation is partitioned into modules? The answer is that different decisions can in fact lead to very different designs. For example, if f(g(x), y) is computed by many F-cells (one for each value of y), then computing g(x) externally enables its value to be computed once and broadcast. Whether this is better than broadcasting x and computing g(x) inside every cell can depend on such factors as the relative data widths of x and g(x), and the relative area costs of computing g versus transferring data.

2.3.2.7 Control Transformations

In general, the purpose of control transformations is to translate software control constructs into forms that correspond directly to hardware constructs, or to move them out of module definitions into the enclosing driver. By doing this repeatedly, problematic constructs can be translated away or extracted into the driver for the whole chip. The chip driver is simply a description of how to use the chip, so it is allowed to contain such constructs.

To illustrate, here are a couple of transformations used in the Pixel Planes example in Section 2.3.2.8.

The first transformation implements complex conditional statements of the form

```
If forall x in S P(x)
then <body>;
```

Note that the If-then construct represents conditional execution rather than a boolean expression: <body> is an action to be performed, not an expression to be evaluated. The transformation implements this conditional by introducing a one-bit Enable register, initialized to True, and using it to compute the condition incrementally. If afterwards the register is still enabled, the <body> is performed:

```
begin
Own boolean Enable;
Enable ← T;
For each x in S
        do Enable ← Enable ∧ P(x);
If Enable
then <body>;
end;
```

The *transformed program* is closer to the hardware level because the complex condition in the conditional has been replaced with a simple flag. This can be implemented by gating the Enable register with the evoke signal for <body>. For example, if <body> is a register assignment, Enable would be gated into the register's write-enable line.

The discerning reader may object to this implementation on the grounds of inefficiency: Although the Enable register may be disabled early on, P(x) will still be computed for every element of the set S. The answer to this objection in the Pixel Planes example is that the transformation is used to implement a conditional performed at every pixel in parallel. By iterating unconditionally, the pixels can share control signals broadcast from a common driver. This illustrates how a program ill-suited to execution on an ordinary sequential processor may represent a hardware-efficient implementation.

A second transformation helps implement begin-end blocks. Consider a call to a procedure whose body is a begin-end block:

```
Procedure F(x);
        begin <stmt_1>; ...; <stmt_n>; end;
end F;

F(<expr>);
```

The transformation extracts the begin-end block into the driver for F by converting the body of F into a case statement and adding a control input to module F that tells it which action to select. Such a case statement can readily be implemented in hardware by combinational logic that decodes the control input and evokes the selected action. The driver is changed into a series of calls to F, one for each statement in the begin-end block:

```
Procedure F(ctrl, x);
        Case ctrl of case_1: <stmt_1>; ...; case_n: <stmt_n>; end Case;
end F;

begin F(case_1, <expr>); ...; F(case_n, <expr>); end;
```

When this transformation and its analog for For loops, described in Section 2.3.2.5, are recursively

applied to a procedure containing nested sequential control constructs, they transform its body into a finite state machine of the sort described in Section 2.3.2.4. The implemented transformation identifies which calls actually use the value of the input x. In those that do not, the pseudo-argument **any** is passed instead. This facilitates input-line-sharing optimizations (not yet implemented). For example, if F has several inputs, but only one is used at a time, they can save area by sharing the same input line.

The above transformation implements *external control*: an external control input tells the module what to do for each call. An alternative is to introduce an internal state register and update it after each call, returning a control signal to notify the caller when the block is completed:

```
Procedure F(ctrl, x);
        own state;
        case ctrl of
                reset:  state ← 1;
                normal: begin in parallel
                                case state of
                                        1: <stmt_1>;
                                        ...;
                                        n: <stmt_n>;
                                end case;
                                if state < n
                                then Return busy
                                else Return done;
                                state ← state + 1;
                        end;
        end case;
end F;

begin
        F(reset, any);
        Do flag ← F(normal, <expr>) until flag = done;
end;
```

Two points should be noted here. First, although the control state is now internal to module F, an input signal is still needed in order to reset to the initial state, and an output signal is used to notify the caller of completion.[2] This is still an improvement, since each signal is only one bit wide, whereas an input line for external control must be $\log_2(n)$ bits wide. Second, this transformation makes use of the two-phase clocking scheme mentioned in Section 2.3.2.1. Since all reads are done in phase 1 and all writes in phase 2, the old value of the state register will be used in phase 1 to select which statement is performed; it will not be updated to the incremented value until phase 2.

2.3.2.8 The Pixel Planes Example

We now show how transformations like those described above can be used to re-derive the Pixel Planes design described in [2]. The Pixel Planes chip displays a color scene from the viewpoint of an observer; it performs color shading and eliminates hidden surfaces. The data input to the chip describes successive scenes consisting of polyhedral objects; a pre-processor segments the objects

---

[2]The output signal could be eliminated at the cost of having the caller keep track of the state separately. This might be worthwhile in cases where a single caller controlled many copies of F in lockstep.

into triangular faces and transforms them into viewing coordinates. The data specify the coordinates and color of each vertex. As Figure 2-1 shows, the problem is to determine what color to paint each pixel on the display screen.

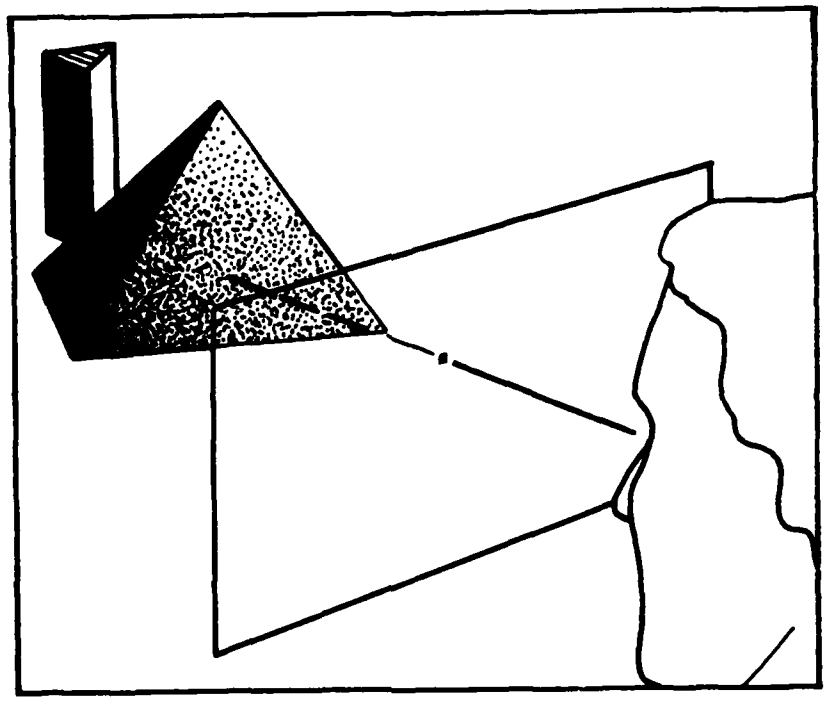


Figure 2-2: The PIXEL PLANES chip performs color shading and hidden surface elimination for a scene composed of polyhedral objects.

The high-level specification for the chip can thus be expressed informally as the following:

```
For each scene
        For each pixel
                Display shaded color of nearest face
```

A formal specification in Gist has been developed, but is omitted here since we are focussing on the transformation from algorithm to circuit. The algorithm can be expressed as:

```
Foreach s in Scenes
        Foreach p in Pixels
                begin
                minZ ← ∞;
                Foreach t in Faces
                do  If Forall e in Edges(t)
                                A_e x_p + B_e y_p + C_e > 0
                        ∧ A_t x_p + B_t y_p + C_t < minZ
                        then        begin
                                minZ ← A_t x_p + B_t y_p + C_t;
                                Display[p] ← A_S x_p + B_S y_p + C_S;
                                end;
                end;
```

How this algorithm works is irrelevant here; we are interested only in its syntactic form. First, notice that it contains several planar expressions of the form $Ax_p + By_p + C$. Second, note the presence of the begin-end blocks and the complex conditional statement "If Forall... $\wedge$... then...."

At the design stage shown below, several design decisions have been made. Some of these are revised by subsequent optimizations, which are omitted here for simplicity.

1. *Allocation*:

   - Use a replicated PixCell[p] to compute pixel color in parallel for each pixel p.
   - Multiplex hardware for computing $Ax_p + By_p + C$ among the different uses.

2. *Data flow*:

   - Compute $Ax_p + By_p + C$ inside PixCell[p].[3]
   - Import the planar coefficients from outside PixCell[p].

3. *Control*:

   - Implement the complex conditional using an Enable register.
   - Use an external control line to select the action performed by PixCell[p].

A key point here is that the transformations used to model these decisions are general, even though the decision to select them instead of alternative transformations is based on low-level considerations specific to the problem at hand. Each transformation automates a particular kind of implementation method , but the problem of selecting among alternative methods requires predicting their low-level consequences and is left to a human expert.

The circuit resulting from these decisions is represented by the following procedure:

---

[3] This would be a poor decision if it were allowed to stand, since using two multipliers for every pixel would be very costly in area. It also pretends that the multiplication is performed in a single clock cycle. In the actual design, the products Ax_p (and By_p) are computed externally and broadcast bit-serially to all pixels with the same value of x_p (or y_p). These refinements are omitted above for simplicity.

```
Procedure PixAry(ctrl, A, B, C);
        For each p in Pixels in parallel
                do PixCell[p](ctrl, x_p, y_p, A, B, C);

        Procedure PixCell[p b[in] Pixels](ctrl, x, y, A, B, C);
                Own Enable, minZ, Z, display;
                Case ctrl of
                        case1:  minZ ← ∞;
                        case2:  Enable ← T;
                        case3:  Z ← A*x + B*y + C;
                        case4:  Enable ← Enable ∧ (Z > 0);
                        case5:  If (Enable ← (Enable ∧ (Z < minZ)))
                                then minZ ← Z;
                        case6:  If Enable
                                then Display ← Z;
                        no-op:  ;
                end Case;
        end PixCell;
end PixAry;
```

The driver for this procedure is as follows:

```
For each s in Scenes in sequence do
        begin
        PixAry(case1, any, any, any);
        For each t in Faces do
                begin
                PixAry(case2, any, any, any);
                For each e in Edges(t) do
                        begin
                        PixAry(case3, A_e, B_e, C_e);
                        PixAry(case4, any, any, any);
                        end;
                PixAry(case3, A_t, B_t, C_t);
                PixAry(case5, any, any, any);
                PixAry(case3, A_S, B_S, C_S);
                PixAry(case6, any, any, any);
                end;
        end;
```

At this stage in the design process, PixAry represents a pixel array that inputs a control code and the coefficients A, B, and C, and broadcasts them to all the PixCells, as well as sending each PixCell its x and y coordinates. The control code tells each PixCell what do to in a given clock cycle; note that case3 is the computation of a planar combination. The nested assignment in case5 is simply a way to specify that the expression Enable ∧ (Z < minZ) is both stored into the Enable register and used as a condition for whether to update the minZ register. The driver tells what sequence of control signals and data to send to PixAry; note that case3, for computing planar combinations, is used several times.

Many of the transformations described in this section are implemented.

- Control transformations are implemented in programs called PRIMITIVIZE and CASIFY. PRIMITIVIZE maps high-level control constructs into lower-level constructs, and CASIFY transforms a "primitivized" procedure into a case-statement body and an external control

driver. Both programs operate without user intervention, since they incorporate fixed control style decisions, e.g., use external rather than internal control.

- Data flow decisions are made interactively by a program called MAKEARGSEXPLICIT. MAKEARGSEXPLICIT systematically traverses a program fragment to be mapped onto a module, identifying sub-expressions that use data external to the fragment and asking the user whether to compute them inside or outside the module. The module ports are determined accordingly.
- The allocation transformations remain to be implemented. A key problem here is to provide the user with a convenient way to refer to the part of a program to which a transformation should be applied. An ideal solution would be to embed the transformations inside a structural or display-based editor. All the transformations will likely be (re-)implemented in the POPART program development environment [7].

2.4 FUTURE WORK

We will continue creating mappings which will provide alternative realizations of Gist's specification freedoms.

These transformations include making backtracking explicit, making nondeterministic choices to reduce or eliminate backtracking, converting backtracking to search, special cases of recursive derived relations (such as transitive closure), maintaining extreme and ordered sets, and various forms of representation selection.

We will also develop a facility for block translating the implementation subset of Gist into two different programming languages. The first will be LISP, and the second will be an algebraic language.

We will then address the problem of interaction between implementations of different constructs. These interactions cause two problems. First, it increases the difficulty of making design decisions because those decisions cannot be considered in isolation. Since the human designer retains responsibility for making the design decisions in our approach, he, rather than the system, must contend with this complexity.

The second difficulty is that since the transformations deal with concrete situations to produce concrete instances and the interacting decisions could be made in any order, then if there are decisions that could interact there must be N↑N transformations. To avoid this combinational explosion we are working on a technique for delaying the effects of making a decision by annotating the constructs being transformed. Subsequent transformations may add further annotations but in general will be insensitive to any existing annotations (the exception is when one annotation is incompatible with another). After the transformation process has been "completed," special transformations can be applied to the combinations that were produced. Thus, the combination sorted-array could produce different algorithms and representations than those produced by the combination sorted-list.

There are several benefits from such an approach. First, by realizing the decisions all at once, the order dependence between those decisions has been removed. This both simplifies the creation of the needed transformations (because they convert a (compound) conceptual entity into an implementation rather than modify one implementation structure to produce a combination of two separate ones) and greatly reduces the number required. Second, by expressing the decision as an

annotation rather than an explicit implementation structure, the comprehensibility of the evolving design is improved. This method provides a way of representing a set of possible instantiations (depending upon other decisions) via a description (i.e., an abstraction).

This mechanism is just the means by which we can introduce those domain specific concepts that we identified as a problem in our VLSI effort (it is just as applicable to the software design domain). Thus, for both domains, we will be identifying the concepts that should use this annotation approach and creating the transformations that handle the combinations of annotations that get generated.

Late next (fiscal) year we will demonstrate the viability of our mapping approach by developing a real working part of some research project via these techniques.

As originally planned, the VLSI mapping effort will closely parallel the Advanced Specification Mapping effort with about a six-month delay so that the methodological problems that arise can be worked out in the more familiar software domain and so that we can concentrate our VLSI efforts in applying that technology to a separate domain.

REFERENCES

1. Earley, J., "High level iterators and a method for automatically designing data structure representation," *Computer Languages* 1, (4), 1975, 321-342.

2. Fuchs, H., and J. Poulton, "Pixel-Planes: a VLSI-oriented design for a raster graphics engine," *VLSI Design*, (third quarter), 1981, 20-28.

3. Lam, M., and J. Mostow, "A transformational model of VLSI systolic design," in *IFIP 6th International Symposium on Computer Hardware Description Languages and their Applications*, Carnegie-Mellon University, May 1983.

4. Paige, R., and J. Schwartz, "Expression continuity and the formal differentiation of algorithms," in *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*, pp. 58-71, Los Angeles, 1977.

5. Paige, R., and S. Koenig, "Finite differencing of computable expressions," *ACM Transactions on Programming Languages and Systems* 4, (3), July 1982, 402-454.

6. Teitelman, W., *Interlisp Reference Manual*, Xerox Palo Alto Research Center, 1978.

7. Wile, D. S., *Program Developments: Formal Explanations of Implementations*, USC/Information Sciences Institute, RR-82-99, August 1982.

3. SPECIFICATION ACQUISITION FROM EXPERTS

Research Staff:
Robert Balzer
Don Cohen
Neil Goldman

Research Assistant:
Brigham Bell

Support Staff:
Audree Beal

Since the SAFE project has been completed, we will review the entire project rather than just its last year.

3.1 PROBLEM BEING SOLVED

This project addressed the difficulty of producing formal software specifications which match the user's intent.

3.2 GOALS AND APPROACH

Our goal was to automatically translate the existing specification mechanism, informal natural language description (MIL-SPEC 490-B5), into a formal specification.

Our approach was to build a natural language understanding system which disambiguated the informal input by incrementally synthesizing alternative formal specifications and rejecting those that did not make sense. This was determined by symbolically executing the hypothesized formal specification to characterize all its possible behaviors and comparing these against a set of well-formedness criteria.

3.3 SCIENTIFIC PROGRESS

We achieved only limited success on our original goal of translating natural language descriptions into formal specifications. A prototype was built which understood and correctly formalized several real-world, albeit simplified, specifications extracted from actual MIL-SPEC 490-B5 specification manuals. It then successfully handled twenty-five perturbations of these examples as a demonstration of its robustness on such small simplified specifications. One of these examples is presented in Figures 3-1 and 3-2.

We believe these results demonstrated the basic feasibility of our approach. The remaining watershed issue was whether this approach could be scaled up to handle large practical-sized specifications (about two orders of magnitude larger than our examples). This required two fundamental changes in our prototype. First, we had to replace the ad-hoc procedurally embedded knowledge used in the prototype with a much more adaptable knowledge base that could be easily augmented as our understanding of the task improved. Second, we had to change from a multi-pass "batch compiler" paradigm to an incremental one in which each new input sentence was understood in the context of, and incorporated into, the formalized specification up to that point.

```
*((MESSAGES ((RECEIVED) FROM (THE "AUTODIN-ASC"))) (ARE PROCESSED) FOR (AUTOMATIC
DISTRIBUTION ASSIGNMENT))

*((THE MESSAGE) (IS DISTRIBUTED) TO (EACH ((ASSIGNED)) OFFICE))

*((THE NUMBER OF (COPIES OF (A MESSAGE) ((DISTRIBUTED) TO (AN OFFICE)))) (IS) (A FUNCTION
OF (WHETHER ((THE OFFICE) (IS ASSIGNED) FOR (("ACTION") OR ("INFORMATION"))))))

*((THE RULES FOR ((EDITING) (MESSAGES))) (ARE) (: ((REPLACE) (ALL LINE-FEEDS) WITH
(SPACES)) ((SAVE) (ONLY (ALPHANUMERIC CHARACTERS) AND (SPACES))) ((ELIMINATE) (ALL
REDUNDANT SPACES))))

*(((TO EDIT) (THE TEXT PORTION OF (THE MESSAGE))) (IS) (NECESSARY))

*(THEN (THE MESSAGE) (IS SEARCHED) FOR (ALL KEYS))

*(WHEN ((A KEY) (IS LOCATED) IN (A MESSAGE)) ((PERFORM) (THE ACTION ((ASSOCIATED) WITH
(THAT TYPE OF (KEY))))))

*((THE ACTION FOR (TYPE-0 KEYS)) (IS) (: (IF ((NO OFFICE) (HAS BEEN ASSIGNED) TO (THE
MESSAGE) FOR ("ACTION")) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS ASSIGNED) TO (THE
MESSAGE) FOR ("ACTION"))) (IF ((THERE IS) ALREADY (AN "ACTION" OFFICE FOR (THE
MESSAGE))) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS TREATED) AS (AN "INFORMATION"
OFFICE))) (((LABEL OFFS1 (ALL "INFORMATION" OFFICES FROM (THE KEY))) (ARE ASSIGNED) TO
(THE MESSAGE)) IF ((REF OFFS1 THEY) (HAVE (NOT) (ALREADY) BEEN ASSIGNED) FOR
(("ACTION") OR ("INFORMATION"))))))

*((THE ACTION FOR (TYPE-1 KEYS)) (IS) (: (IF ((THE KEY) (IS) (THE FIRST TYPE-1 KEY ((FOUND)
IN (THE MESSAGE)))) THEN ((THE KEY) (IS USED) TO ((DETERMINE) (THE "ACTION" OFFICE))))
(OTHERWISE (THE KEY) (IS USED) TO ((DETERMINE) (ONLY "INFORMATION" OFFICES)))))
```

Figure 3-1: Actual input for message-processing example

Each of these two changes generated a major development effort (Hearsay-III and Gist described below) from which we never emerged. The refocusing of our efforts on these two tasks, as necessary precursors of a scaled-up SAFE system, led to the creation of separate projects for them and the termination of SAFE itself.

This still leaves the question of whether the SAFE approach can be scaled-up to handle practical sized specifications. At some point in the future, when the formal groundwork has been more fully prepared, we would like to revisit this question.

In addition to the progress on our original objectives described above, we accomplished the following:

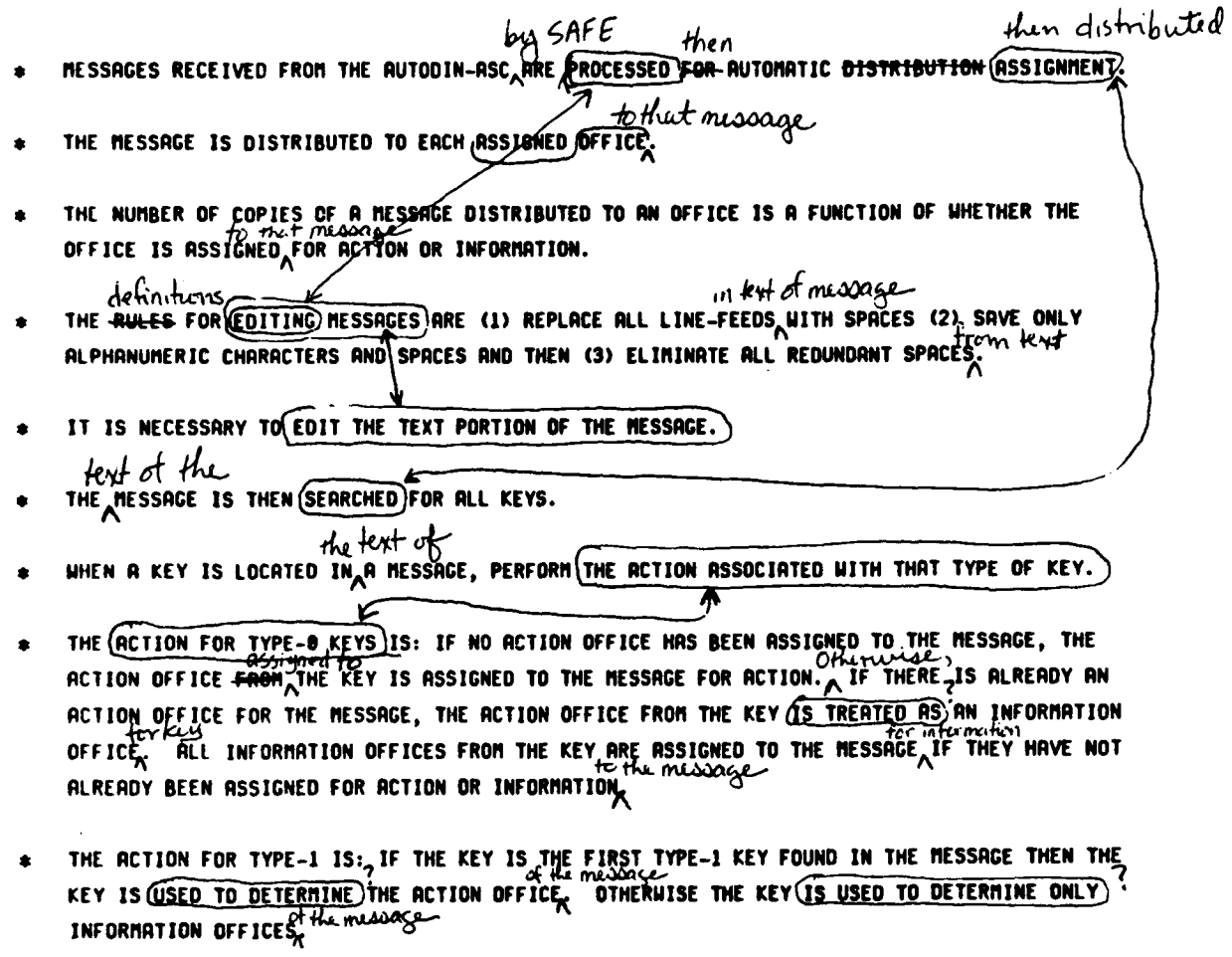


Figure 3-2: Specification deficiencies of message processing example (by conventional programming standards)

1. Formal Specifications

   We developed an adequate formal language (Gist) to express specifications. This language differs from other specification languages in its ability to formally express constructs commonly found in informal specifications (rather than relying on provability criteria), its database viewpoint, its commitment to operationality, its tolerance of incompleteness, and its avoidance of implementation issues. This ARPA-developed language is the basis for a specification validation project sponsored by RADC, a transformation-based development system sponsored by NSF, and a new ARPA project that is developing mappings from these specification constructs into implementations. In addition to these roles, it also was the output language for SAFE. Gist thus occupies a central position in our efforts.

2. Among the unique features of this language are

   a. Semantic use of constraints. Rather than treating constraints as redundant specifications which could be used for checking consistency, Gist uses them to restrict the acceptable interpretations for nondeterministic constructs. Only those choices which don't violate constraints now, or in the future, are acceptable. This

use of constraints provides an extremely powerful formal specification construct which enables us to describe a process behaviorally, rather than algorithmically, in a way closely paralleling the informal way we resolve ambiguity in natural language.

b. Database view including inference. Gist represents the current state of a modeled system as a global database of relations among typed objects. Actions cause these relationships to change. The state of the database and of the objects within it is accessed through a uniform information extraction (query) language. This language locates objects through descriptions of their relationships to other objects. Furthermore, it hides the distinction between explicit and implicit information by including inference within the information-extraction language. Since a major portion of any computing system is concerned with spreading the effects of changes that occur, the self-organizational aspect of the automatic inference mechanism allows this portion of the system to be completely suppressed from the specification.

c. Historical reference. In addition to providing a uniform method of extracting information from the current state, Gist provides the same capability for all previous states of the system. This means that when historical information is needed, it can be directly referenced in the specification. This direct reference avoids having the specifier invent and maintain an auxiliary data structure, and closely approximates the informal mechanisms used in natural language.

3. Hearsay-III

We designed and implemented an artificial intelligence system for developing expert systems, called Hearsay-III, as a base for the revised SAFE system. It is currently being used by SAFE and two other ISI projects. Hearsay-III is a domain-independent tool for constructing knowledge-based systems. It is appropriate for large, complex problems, for which many interacting sources of knowledge are required, specifically including those whose solutions are synthesized incrementally from partial solutions, and excluding simple "diagnosis" problems in which a solution is chosen from among an easily enumerated set.

The Hearsay-III system was designed to provide facilities to represent, compare, and pursue competing, incrementally constructed problem solutions. It is intended to support experimentation with long-term, large-system development. In particular, the design goals of the Hearsay-III system focus on development and debugging of theories of expertise in an application domain; its main goal is to allow for study of a domain, rather than to construct an expert performance system.

Hearsay-III adopts the heritage established by the Hearsay-II speech understanding system. Many of the mechanisms available to the system builder in Hearsay-III are generalizations of mechanisms present in Hearsay-II. However, Hearsay-III is domain-independent; it has no specific knowledge of speech understanding or any other domain at the outset.

Hearsay-III provides primitives that encourage a certain architectural style in the target system. This architecture includes the following features:

a. A structured workspace (called a blackboard) encouraging a hierarchical representation of data organized by various levels of abstraction.

b. Condition/action rules (called knowledge sources) which react to situations represented on the blackboard, and whose action components can modify the blackboard. Condition satisfaction and action invocation are independently controllable events. The action component of a knowledge source is arbitrary Interlisp code, and thus all of its mechanisms for composition and abstraction can be used to raise the level of action expression above that of the Hearsay-III system primitives.

c. A metalevel problem solver to which the same facilities are available (i.e., it can use a scheduling blackboard and scheduling knowledge sources). The task of this problem solver (called the scheduler) is to resolve conflicts among the potentially

large number of knowledge sources whose conditions have been satisfied. The scheduler chooses the next knowledge-source action to execute. The ability to construct a knowledge-based scheduler that reasons about pending knowledge-source activations provides an important step in the direction of separating competence knowledge from performance knowledge in the target system.

d. Facilities for explicitly representing decisions as data. Choices being considered by a Hearsay-III problem solver can be encoded on the blackboard. Decisions represented in this way can satisfy knowledge-source patterns, and can be modified by knowledge-source actions. Decisions therefore can be reasoned about in much the same way as other objects appearing on the blackboard.

e. The blackboard and all publicly accessible Hearsay-III data structures are represented in a relational database (AP3 described below). The typed relational capabilities provided by AP3 are available for directly modeling the application domain.

f. A context mechanism, which allows independent pursuit of competing approaches to solving a problem. The context mechanism allows several different versions of the blackboard, representing competing partial problem solutions considered in separate worlds, to be examined and manipulated independently by the knowledge sources.

4. Symbolic Evaluation of Formal Specifications -

   This capability formed the basis of the spinoff Specification Validation effort (see below) and is more fully described there.

5. Use of Symbolic Evaluation for Understanding -

   This is our main scientific result on SAFE. We demonstrated that disambiguation of informal specifications could be significantly aided by using the dynamic context in which a piece of information was to be used as the basis for disambiguation rather than the syntactic context in which it arose. This dynamic context is provided by symbolic evaluation.

6. AP3 -

   We developed a Planner-like system which stores information associativity, performs (user specified) inference on that information, checks it for type compatibility and against user supplied constraints, and invokes user supplied demons when their firing patterns are matched. This facility is our knowledge representation system and has been used as the implementation basis for most of our work including SAFE, Gist, Hearsay-III, and the symbolic evaluator.

3.4 IMPACT

In addition to the achievements cited above, the SAFE project served as the incubator for four other ISI projects which address closely related aspects of software technology:

1. Transformational Implementation: formalization, mechanization, and automated support of the software development process through transformations.
2. Mapping: development of high level transformations to deal with the freedoms afforded by the specification language (Gist); investigation of the generality of the development paradigm by applying it to the related domain of VLSI design (see Chapter 2).
3. Specification Validation: ensuring before implementation that the specification meets the user's intent by testing entire classes of test cases simultaneously (via symbolic evaluation) and explaining the behavior produced in natural language.
4. Information Management: development of an integrated user evolvable computing

environment based on specification techniques and their support via transformational implementation.

Despite the significance of the above achievements and spinoff efforts, perhaps the most significant payoff of the SAFE effort was the recognition of the inherent flaws in the current software paradigm and the creation of a proposed alternative which has formed the context and integrating force for all the above efforts and which has been incorporated into both DoD's Software Initiative and its Knowledge Based Software Assistant project.

A short summary of this view of how software could, and should, be developed and maintained follows.

Need for an Alternative Software Paradigm

The existence of a software problem has long been recognized. Yet, attempts to date to resolve this problem have yielded only modest gains arising primarily from use of higher level languages and improved management techniques. These gains, which by the most optimistic estimates have resulted in far less than an order of magnitude gain over the last 15 to 20 years, have in no way kept pace with the astounding 1000-fold increase that has occurred in hardware productivity over the same period. As the hardware revolution apparently will continue at this pace for at least the rest of this decade, it is clear that the utility of computers will be limited by our ability to construct, maintain, and evolve software systems.

From where will the elimination of these problems and needed improvements in software productivity arise? Continuation of existing efforts to improve the current software paradigm, broadly characterized as Software Engineering, will undoubtedly yield further incremental improvements more or less commensurate with those previously obtained, subject to the law of diminishing returns.

Unfortunately, this software paradigm, which arose in a drastically different era when machines rather than people were expensive and in limited supply, is fundamentally flawed in two ways which preclude qualitative improvements. First, the process of programming (the conversion of a specification into an implementation) is informal and largely undocumented. It is just this information, and the rationale behind each step of this process, that is crucial, but unavailable, for maintenance (all activity--including debugging--directed at changing the functionality of the system to get it to correspond to the user's evolving intent). Second, maintenance is performed on the source code (i.e., the implementation). All of the programmer's skill and knowledge has already been applied in optimizing this form (the source code). These optimizations spread information (take advantage of what is known elsewhere) and substitute complex but efficient realizations for abstractions. Both of these effects exacerbate the maintenance problem by making the system harder to understand, by increasing the dependencies among the parts, and by delocalizing information.

In the current era of cheap hardware and expensive people, it is clear that automation is the only lever that will qualitatively improve software productivity. System design, implementation and maintenance are all knowledge intensive activities. But the current paradigm, by dealing only with the products of these processes rather than the processes themselves, precludes the use of automated tools to aid those processes.

Thus, an alternative software paradigm, based on representing and supporting, via automated tools, the knowledge intensive processes of requirement specification, system design, implementation, and maintenance is required.

Description of Goal: Alternative Software Paradigm

The automation based software paradigm of the future will provide an integrated set of tools that directly support human developers in the requirements specification, system design, implementation, and maintenance processes. It will be characterized by the fact that the entire history of system evolution (all four of these processes), as directed by the developers, will occur, and be recorded, within the integrated environment. This history will provide the "corporate memory" of the system evolution and will be used by an active environment to determine how the parts interact, what assumptions they make of each other, what the rationale behind each evclutionary step was (including implementation steps), how the system satisfies its requirements, and how to explain all of these to the developers of the system.

This "knowledge base" must be dynamically acquired as a by-product of the development of each system. It must include not only the individual manipulative steps that ultimately lead to an implementation, but also the rationale behind those steps. Both pieces may have to be explicitly stated by the developers. Alternatively, explicit statement of the rationale by the developer may enable the environment to select and perform a set of manipulations that achieve that objective for the developer.

We have described two major differences between the automation based software paradigm and the existing software paradigm--the role of the history of system evolution and the automation it will enable--but we have not yet described the changes that will occur in the software process itself.

First and foremost among these changes will be the emergence of formal specifications as the *linchpin* around which all of the development revolves. In contrast to current practice, in which specifications only serve as an informal description of functionality and performance, which implementors and testers use as a guideline for their work, the actual implementations will be derived from the formal specifications. This will occur via a series of formal manipulations, selected by the developer and applied by the automated tools of the environment, which convert the descriptions of WHAT is to happen to descriptions of HOW it is to happen efficiently. To the extent that these formal manipulations can be proved correct, the validation paradigm will be radically altered. Rather than testing the resulting implementation against the (informal) specification, its validity will arise from the process by which it was developed. That is, the development will constitute the proof of correctness.

But testing in the current paradigm supports more than just the comparison of the implementation with the (informal) specification, it also provides the means, through hands on experience with the working implementation, to compare actual behavior with the user's intent. Often, if not usually, mismatches are detected and the implementation must be revised.

However, in our automation based software paradigm, because the specification is executable--i.e., it is formal and includes the full functionality and performance criteria, the specification itself can, in principle, be used as a prototype to test (i.e., compare) the user's intent to the formal description. Furthermore, some form of automatic or highly automated "compilation" must be created to provide reasonable (though not production quality) efficiency for running test cases.

Thus, the formal specification will be used as a prototype of the final system (i.e., tested), and this prototype will be developed into that final production implementation. As opposed to current practice in which prototyping is the exception, it will become standard practice in the new software paradigm because of its ready availability (via automatic or highly automated "compilation"). In fact, most

systems will go through several prototyping/specification-revision cycles before implementation is undertaken.

But systems are not static, even ones that via prototyping match the user's intent and that via automated assistance are validly implemented. They evolve because the user's needs evolve, at least in part in response to the existence and use of the implemented system. Today, such evolution is accomplished by modifying (maintaining) the implementation. In the automation based software paradigm, such evolution will occur by modifying (maintaining) the formal specification (rather than the implementation) and then reimplementing the altered specification by modifying and "replaying" the previously recorded implementation *process* (the sequence of formal manipulations that converted the specification into the implementation).

This represents another major shift from current practice. Rather than consisting of attempts to "patch" the optimized implementation, the maintenance activity will much more closely parallel the original development. That is, first the specification will be augmented or revised (just as it is modified as a result of feedback from the prototyping/specification-revision cycle). Such modifications should be much simpler because they more closely approximate the conceptual level at which managers understand systems and for which their perception is that such modifications are trivial (it is the highly intertwined, delocalized, and sophisticated realizations that make modification of implementations so difficult). The second step in maintenance is reimplementing the specification. This is another situation in which recording the development process provides leverage. Rather than recreating the entire implementation process, those aspects of the previous development that either must be altered because they no longer work, or should be altered because they are no longer appropriate (i.e., they lead to an inefficient implementation), will be identified by the developer with the help of automated tools in the environment and modified appropriately. Then, this altered development will be "replayed" by the automated tools to obtain a new implementation. To the extent that automation has been used to fill in the details of the implementation process, as described earlier, the need to modify the development will be lessened as these problem solving tools will automatically adjust those details to the new situation. In any case, the effort required to reimplement a specification is expected to be a small percentage of that required for the initial implementation, which in turn is expected to be a small percentage of that required for current conventional implementation.

Thus, in the automation based software paradigm the effort (and corresponding time delay) required both for implementation of the initial specification and especially for maintenance of that specification will be greatly reduced. This will allow that saved energy to be refocused on improved specification (matching the user's intent), on increased functionality in the specification (because implementation costs and complexity restrictions have been mitigated), and on increased evolution of that specification as the user's intent changes over time (at least in part because of the existence of the implementation system).

This will produce one of the most profound effects of the automation based software paradigm. Systems will be larger, more integrated, longer lived, and will be the result of a large number of relatively small evolutionary steps. Software systems will finally reach their potential of remaining "soft" (modifiable) rather than becoming ossified, hardware-like, with age.

Evolution will become the central activity in the software process. In fact, rather than being limited to maintenance after the initial release, it will also become the means by which the "initial" specification is derived. The current "batch" approach to specification in which the specification emerges full-blown all at once (often as a 300 page tome) will be replaced by an "incremental"

approach in which a very clear formal specification is successi ely elaborated by the developer into the "initial" specification. These elaborations will occur via semantic manipulations (rather than "text editing") which capture the various types of elaboration (e.g., adding exceptions to a "normal" case, augmenting the functionality of a process, and revising an earlier description). Thus, specifications will undergo a development just like the implementations they describe. Maintenance of the specification, whether after initial release of the implemented system or as part of the elaboration of the initial specification, will occur by modifying this development structure rather than by "patching" the specification.

To summarize, the automation based software paradigm will differ markedly from the existing paradigm. The basis for this new paradigm will be capturing the entire development process (the identification of requirements, the design of the specification, the implementation of that specification, and its maintenance) and supporting it via automated tools. The development process will revolve around the formal specification. Tools will exist to develop an "initial" specification incrementally via a series of formal manipulations, to test the specification against the user's intent by treating it as a prototype (because the specification is executable), to develop an efficient implementation from that specification via further formal manipulations (which constitute its proof of correctness), and to maintain the system by further developing the specification and its implementation and then replaying that implementation development. This will result in evolution as the central development activity and will produce systems that are longer lived, larger, more highly integrated and which remain pliable to further modification as user needs themselves evolve.

4. VLSI--SERVICES AND RESEARCH

Research Staff:
Danny Cohen
Yehuda Afek
Ron Ayres
David Booth
Joel Goldberg
George Lewicki
Lee Magnone
Lee Richardson
Barden Smith
Vance Tyree

Support Staff:
Victor Brown
Victoria Svoboda
Sharyn Brache

4.1 INTRODUCTION

The VLSI project is aimed at the advancement of a low-cost, fast turnaround VLSI device fabrication service to support the geographically distributed DARPA and NSF supported VLSI research communities, which have no direct access to VLSI fabrication, but have access to computer communication networks.

A special effort has been made to implement a VLSI fabrication service called MOSIS (MOS Implementation System). MOSIS has high-level automatic-access control, and authentication and protection mechanisms. The motivation for this service is to provide an "almost interactive" VLSI chip design environment with a design-to-packaged VLSI device turnaround time of a few weeks. We believe this service is causing a revolution in system architecture research as dramatic as the introduction of interactive computation was to programming.

The current major thrust of the MOSIS operation is the support of advanced CMOS with small feature size.

The two most exciting projects supported by MOSIS in the reporting period are RISC [1] and the Geometry Engine.

4.1.1 Problem Statement--Fabrication

Fabrication is the entire process of converting designs in machine readable form (Caltech Intermediate Form (CIF) in this case) into actual devices. It includes the grouping and placement of projects on dies and wafers, converting CIF into the target pattern generation MEBES format, mask generation, wafer fabrication, fabrication quality testing by wafer probing, device separation (scribing or sawing), packaging, bonding, and finally distribution to the designers.

- The VLSI design communities of ARPA and NSF need fabrication capabilities in order to investigate the design methodologies and architectures appropriate to VLSI where gate-count will exceed a million gates per device.
- These fabrication capabilities must be of sufficiently low cost to be affordable by the research community.

- The turnaround time must be fast enough to support semi-interactive research (a few weeks, at most). Design and fabrication must be decoupled in order to achieve fabrication line independence similar to device-independent programming.
- Many fabricators must be involved in order to ensure the continuous availability of custom VLSI fabrication.
- Since the designers are not involved in the interface with the fabrication vendors, it is essential that a new testing methodology, which does not depend upon testing the circuits submitted by the designers, be used to evaluate the quality of the fabrication.
- Design rules and CIF for expressing their patterns are available now for nMOS, CMOS/bulk and CMOS/SOS. Since the move to new technologies and smaller geometries is ongoing, these capabilities are continuously supplemented.
- Wafer probing is necessary to assure the reliability of the fabrication operation.

4.2 TECHNICAL APPROACH AND GOALS

The work in the VLSI project is focused on the fabrication services for the VLSI research communities of DARPA and NSF, and the required support.

4.2.1 The Fabrication Service

After a period of debugging, the MOSIS system became operational in January 1981. The support of nMOS fabrication at 5 and 4 micron feature size became the standard routine operation. The support of 3 micron nMOS and of CMOS is approaching this state, too.

The fabrication service area of the VLSI project was scoped to achieve the following:

- Development, operation, and maintenance of MOSIS for use by the VLSi research communities of DARPA and NSF. The MOSIS system accepts VLSI designs expressed in CIF and handles all the fabrication steps needed in order to provide the designers with actual chips. The major components of the MOSIS system are

  - interaction with the designers,
  - handling of their CIF files,
  - communication over either the ARPANET or TeleMail,
  - placement of projects on dies, and dies on wafers,
  - matching MOSIS design rules to specific vendor's design rules, addition of alignment marks, critical dimensions, and test devices.
  - fabrication of E-beam masks (via subcontract),
  - wafer fabrication (via subcontract),
  - wafer probing and data analysis,
  - generation of bonding maps,
  - wafer sawing, die packaging, and bonding (via subcontract),
  - chip distribution.

- As part of running this service for the DARPA and NSF communities, the MOSIS crew interacts with various designers in order to help them use the system, receive important information, etc. The crew also interacts with the various vendors in order to establish many of the parameters required for each run. As part of its mission, the MOSIS crew is continually seeking new vendors and encouraging them to offer custom VLSI fabrication service.

- A major activity of MOSIS is the development of a testing methodology required for efficient decoupling of the design from the fabrication process. These activities are described in following sections.
- Improve the design of the MOSIS system to expand its capabilities, especially in access control, budget control, and management information.
- Implement the MOSIS system to address those concerns that become important in the context of handling tens of thousands of designs, such as efficiency, impact on system resources, and the interaction with the file archive system.
- Interface to as many mask houses and fabrication lines as is practically possible, as part of promoting the "broker/foundry" concept.
- Coordinate the development of testing procedures required for quality control of the MOSIS operation and keep databases for use at any time in yield data analysis.
- Control the performance of both testing and packaging as required for the fabrication service offered by MOSIS.
- Design cell libraries to support CIF entries, provide access control to the library, and automatically distribute information about the content of the library and its usage.
- Maintain the design library and provide the necessary support for it.

4.3 SUMMARY OF PROGRESS

4.3.1 The Acceptance of MOSIS by the Industry

MOSIS seems to have been readily accepted by the industry. The best testimonial for this acceptance is the many instances in which semiconductor companies use the MOSIS devices (wafers and dies) as part of their publications or advertisements, usually with our permission.

Many publications, especially those in various electronics fields, often mention MOSIS and display its devices, as do publications in other fields. For example: *Aviation Week and Space Technology* [23 February 1981, and 1 June 1981], *Business Week* [14 June 1982], *Electronics* [20 October 1982], *Semiconductor International* [June 1982], *Military Electronics* [October 1982] and *VLSI Design* [July/August 1982 and September/October 1982].

A MOSIS wafer from run M24D-Delia was featured on the cover of the July/August 1982 issue of *VLSI Design* (Fig. 4-1) as an example of the capabilities offered by ZyMos. In the same issue ComDial uses another MOSIS wafer to demonstrate their capabilities, and AMI displays a picture (Fig. 4-2) taken of the Geometry Engine die from the M17M run, showing the old MOSIS test strip as an example of proper user testing. An article in *Semiconductor International* [June 1982] demonstrates the Customer Owned Tooling (COT) approach by showing both M19P (Fig. 4-4) and M11E wafers (Fig. 4-3). The October 20, 1982, issue of *Electronics* shows a MOSIS wafer (Fig. 4-5) as an example for the Silicon Sharing concept. An article about silicon compilers, in the September/October 1982 issue of *VLSI Design*, shows MOSIS devices M24DJ1 and M24DJ2 (Fig. 4-7) as examples.

CSI Technology Services routinely use MOSIS wafers as an example of their ability to process non-standard wafers. Pictures of MOSIS wafers may be found on sales flyers under various headings like "Hughes Tweaks Process for Solution," "ATARI Cuts Prototyping Time 77%," and "Bendix Beats Tight Deadlines" (Fig. 4-6).

Figure 4-1: M24D wafers on the cover of *VLSI Design*

4.3.2 The User Interface

The MOSIS user interface is based entirely on an exchange of electronic mail messages over the ARPANET or TeleMail between the user and MOSIS. Unless a user specifically asks to communicate with a person (usually only to handle a special request), the entire message exchange is handled

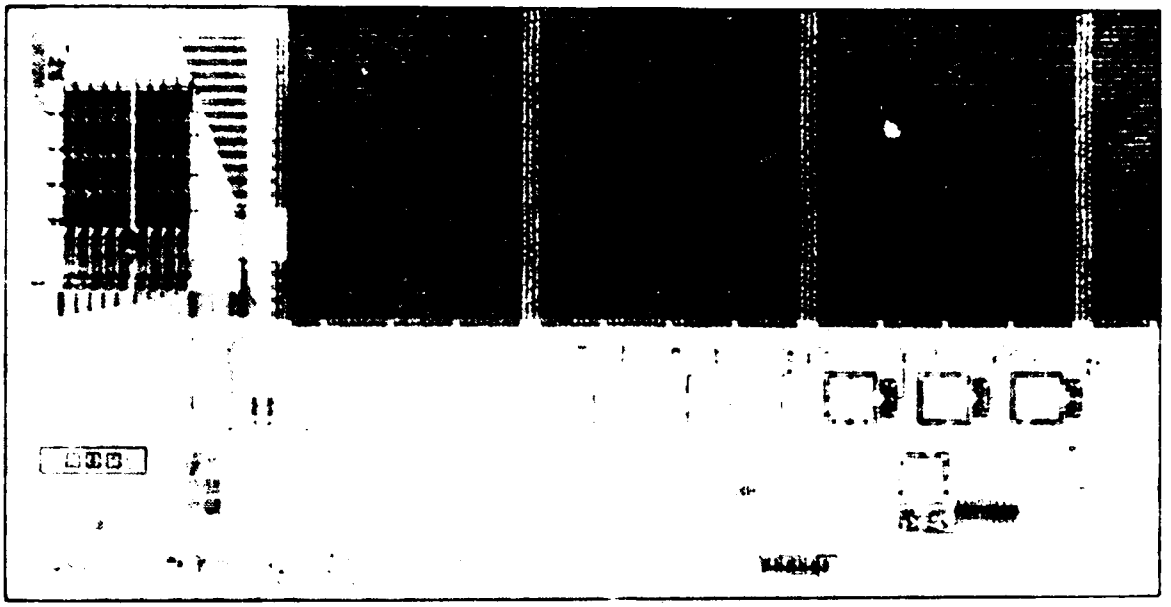


FIGURE 2. A "test strip" which contains the critical dimension should be included on each die in a multi-project wafer.

Figure 4-2: The old MOSIS test strip

automatically by MOSIS. This user interface works so smoothly that several users were under the impression that the interchange was handled manually by the MOSIS crew.

User interaction with the system has the following steps:

1. Approval of the user privilege to use MOSIS services
2. Information requests
3. Definition of new projects
4. Submission of CIF for fabrication
5. Status and progress reports from MOSIS
6. Distribution of fabricated devices (chips)

In addition to accepting CIF as the pattern definition format, MOSIS can accept designs in the MEBES format and merge them with CIF-submitted designs.

MOSIS uses several mechanisms to protect the proprietary nature of design information. These include passwords both for the individual designers and for each project, so that users who cooperate in working on certain projects do not have to provide each other access to all of their other designs.

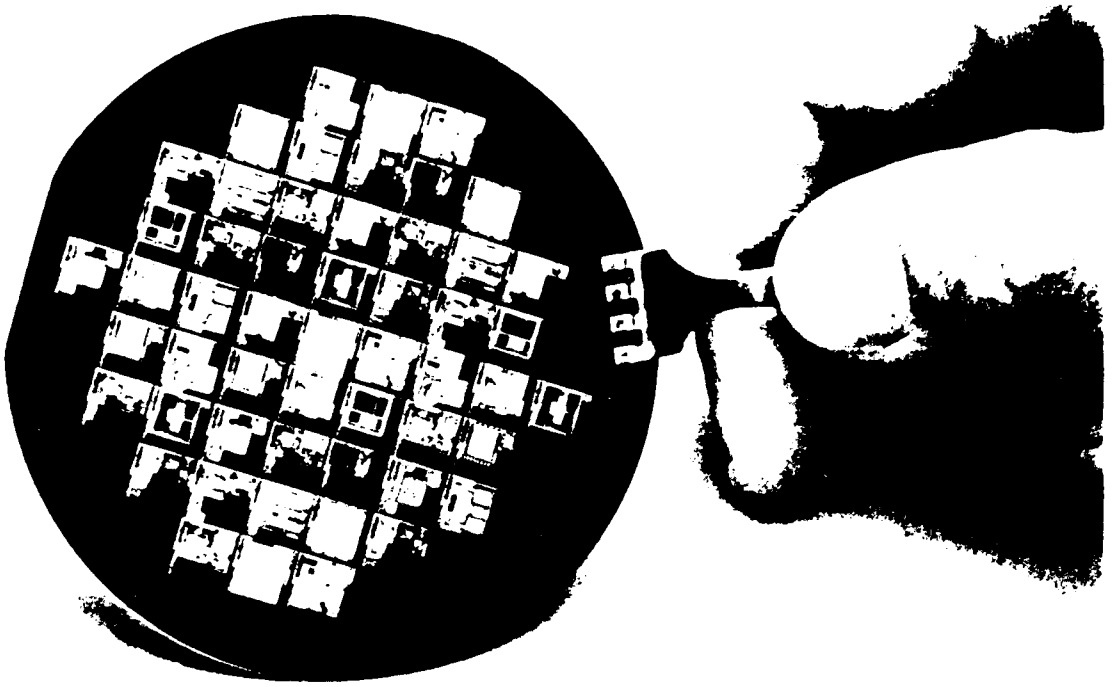


Figure 4-3: The M11E wafer

4.3.3 The Fabrication Interface

The MOSIS interface to the fabrication facilities has several aspects. The most important interaction is with the wafer fabrication facilities. Depending on the various specifications of the particular fabricator, MOSIS may have to adjust several parameters, such as geometric compensation (bloats, shrinks, alignment marks, critical dimension and test devices), and generate the appropriate set of masks.

MOSIS shields the designers from many details involved in the fabrication process. We are very proud that most MOSIS users do not even know about these details and that they need not know about them in order to obtain fabricated ICs. All of these details are handled by the "control protocol" of the MOSIS interface to the fabrication facilities.

The bulk of the interface, "data protocol," is a set of MEBES tapes. These are used by a mask house to produce a set of glass masks (by using an E-beam pattern generator), which are required for

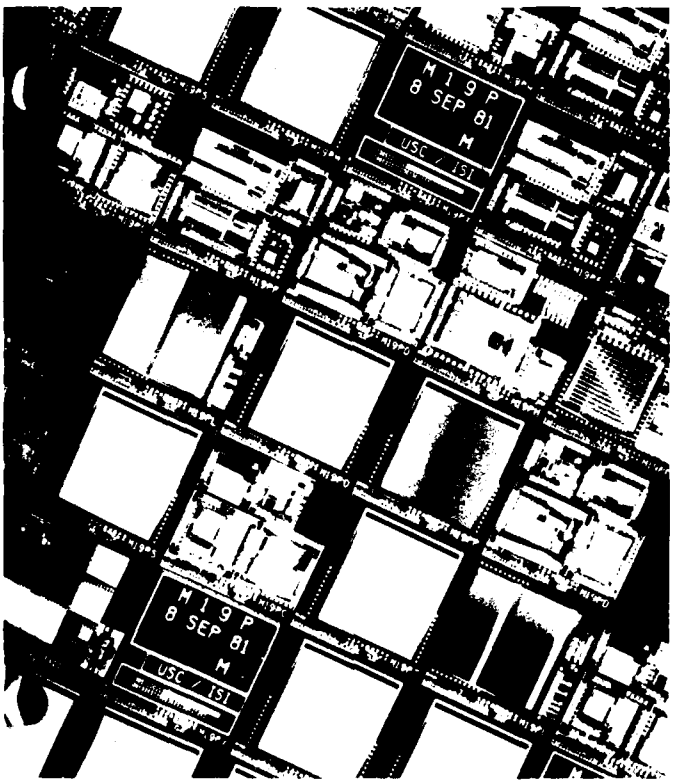


Figure 4-4: The M19P wafer

fabrication. The next portion of the fabrication interface generates a set of bonding maps (automatically produced by MrBill, the MOSIS geometry processor) needed to turn chips into usable devices.

MOSIS adds critical dimension and alignment-marks to each die. Unless the vendor insists on the use of its own devices, MOSIS uses the MOSIS standard critical dimensions (tuning forks and dual-polarity crosses) and dual-polarity alignment-marks (Figs. 4-9 and 4-12).

In addition, MOSIS adds to each wafer, inside one of the MOSIS drop-ins, its CKSIZE device (Fig. 4-13) which is used for optical measurements of the actual line widths both on the E-beam masks and (for the visible layers only) also on the wafers. This measurement is used for verifying the actual bloats/shrinks which occur in the fabrication process. The resolution of this measurement is twice the resolution used in the E-beam writing of the masks (0.25 micron in our case).

4.3.4 MrBill, the Geometry Processor

All the IC-specific processing of the MOSIS service is done by an ICL program called MrBill. MrBill accepts individual projects submitted over the ARPANET or TeleMail to MOSIS. Each design is represented in CIF.

MrBill reads in CIF files, checks their validity, and converts them into an internal representation.

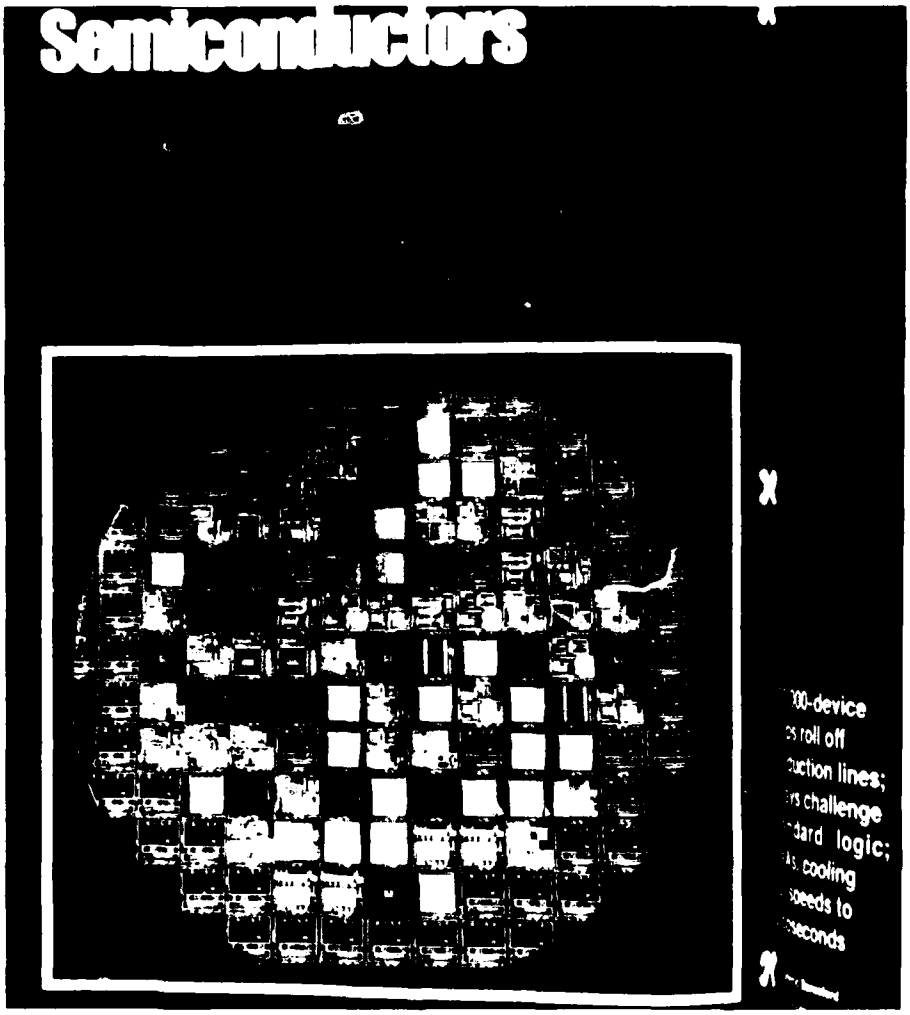


Figure 4-5: A MOSIS wafer as an example of the Silicon Sharing concept

MrBill then "packs" the projects onto individual dies (chips) in such a manner that no project interferes with any other project. It is this packing that allows the costs of fabrication to be distributed among many projects, rendering a small cost per project and a more efficient use of silicon.

Finally, MrBill translates each chip into MEBES format. It is this format that the mask fabricator's E-beam mask machine understands. The geometric processing is complete at this point.

MrBill's database also produces bonding diagrams for the people who package our chips. A bonding diagram shows the bonders how to connect the chip to the pins on the package.

MOSIS quality assurance

In order to evaluate the quality of the MOSIS fabrication runs, it is necessary to have a set of test vehicles that can be placed on each wafer or die. In the MOSIS runs, test vehicles serve as a common ground between designers and fabricators. A designer executes his design on the assumption that some basic elements (such as transistors and interconnects) behave in a particular way and on the expectation that a large number of elements can be defectless. To verify his design, he submits it to a

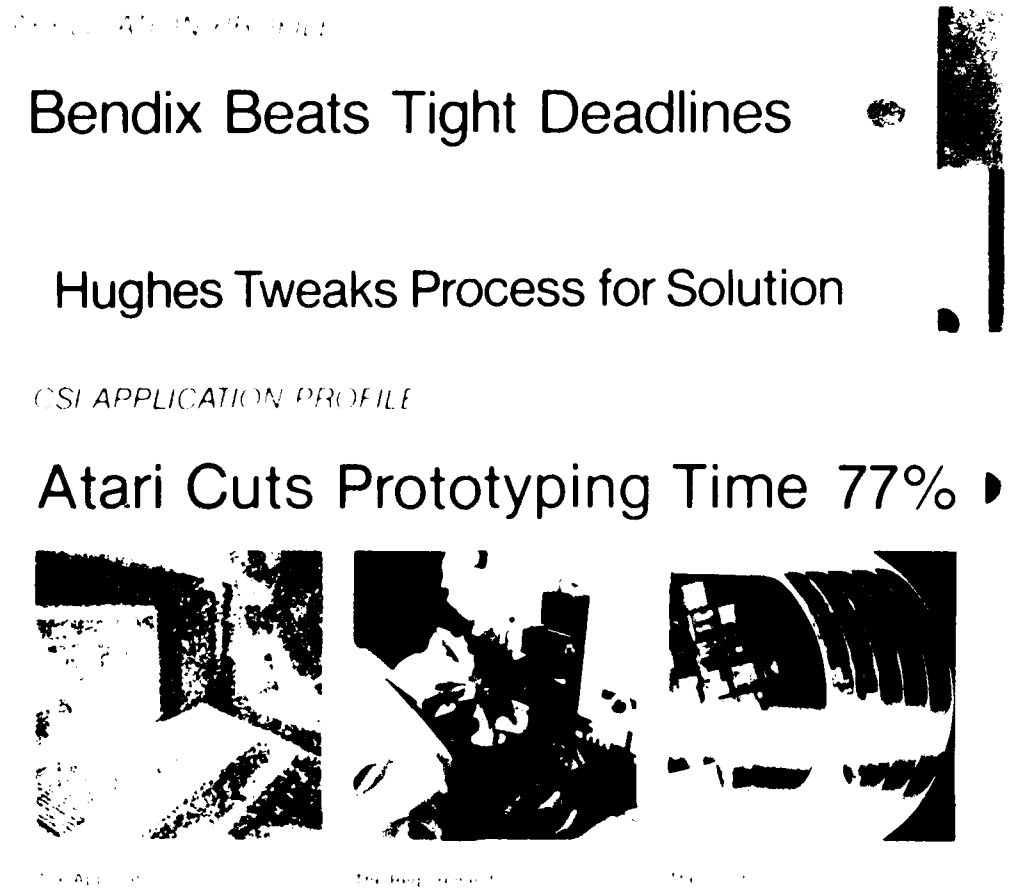


Figure 4-6: Application flyers of CSI showing MOSIS wafers

MOSIS run. Before investing energy into verifying his design by testing the operability of a fabricated circuit, he should be assured that these assumptions regarding the behavior of the basic elements are correct. Measurements on test vehicles either on the same die or on the same wafer containing the designer's circuit are the only practical way to provide such assurances. Such test vehicles also benefit the fabricator. The quality of his product can be more accurately and fairly judged on the operability of standard test circuits than on the operability of an unproven and perhaps badly designed circuit submitted by a designer.

Such test vehicles are also of considerable value to the broker bringing designers and fabricators together. It allows that broker to make meaningful assessments of the capabilities of various fabricators.

4.3.5 Test Vehicle Developments for MOSIS

There are basically three types of test structures in use for evaluation of the fabrication runs:

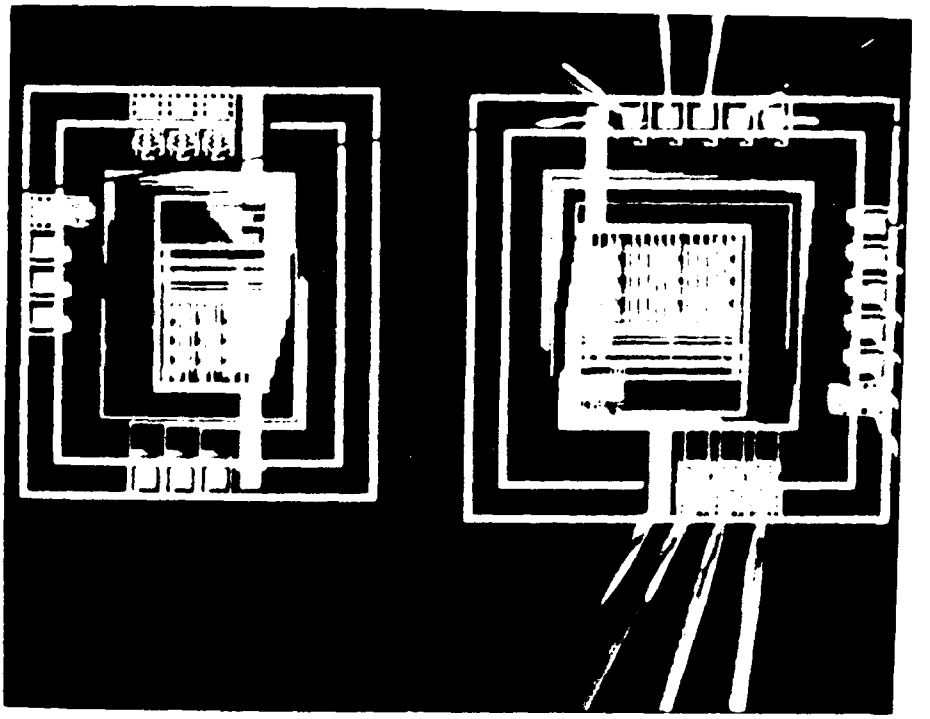


FIGURE 3 These small chips (4-bit parallel in out left right shift register on the left, 4-bit resettable modulo-n counter on the right) were designed by an early version of the MacPitts system. Larger devices have yet to be fabricated

Figure 4-7: MOSIS devices M24DJ1 and M24DJ2

1. Test Strip: This test structure is placed on every die to enable evaluation of critical fabrication parameters spatially (Figs. 4-9, 4-10, and 4-11).
2. Random Fault Structures (RFS): This is a collection of fundamental circuit features placed in a test die ("Drop-in") to enable rapid evaluation of the reliability of fabrication. It is possible to use the RFSs to separately evaluate the reliability of fabrication of each conducting layer and interconnection method in various topological configurations that commonly appear in circuit designs. This information is valuable when analyzing failures in the yield monitors.
3. Yield Monitors: This is a drop-in test structure that is implemented to evaluate yield of circuits designed using the standard MOSIS geometric design rules. Currently the test structure consists of a 4096-bit random access memory (canary), fondly referred to as "the world's slowest 4-K RAM," plus a relatively small (compared to the canary) dynamic storage device to provide yield information about dynamic circuitry.

The Test Strip consists of eight basic DC parametric test structures that are the primary source of information used to select wafers from which dies are cut for packaging and distribution to the community. The basic parametric test structures include

- Large Transistors for measurement of threshold, KP, and GAMMA.
- Variable channel length and variable channel width transistors for measuring short/narrow channel effects upon the threshold and other parameters (Figs. 4-14 and 4-17).

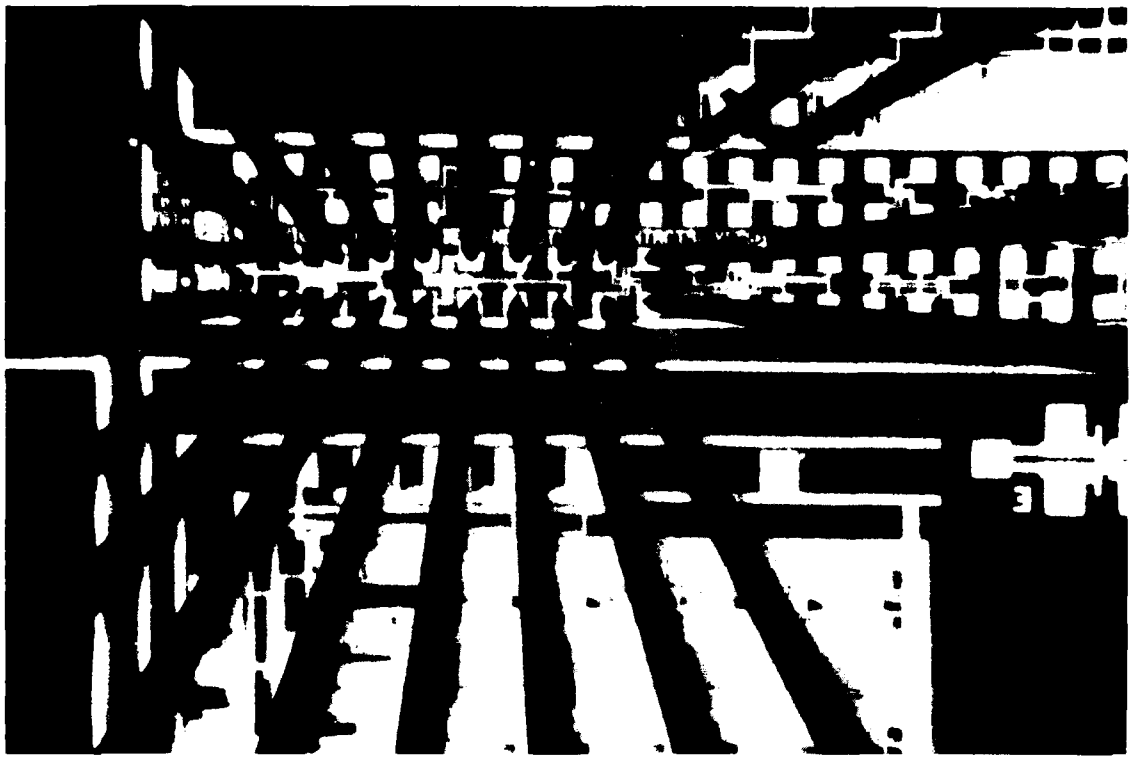


Figure 4-8: Probing of the standard MOSIS test strip (2x10 pads)

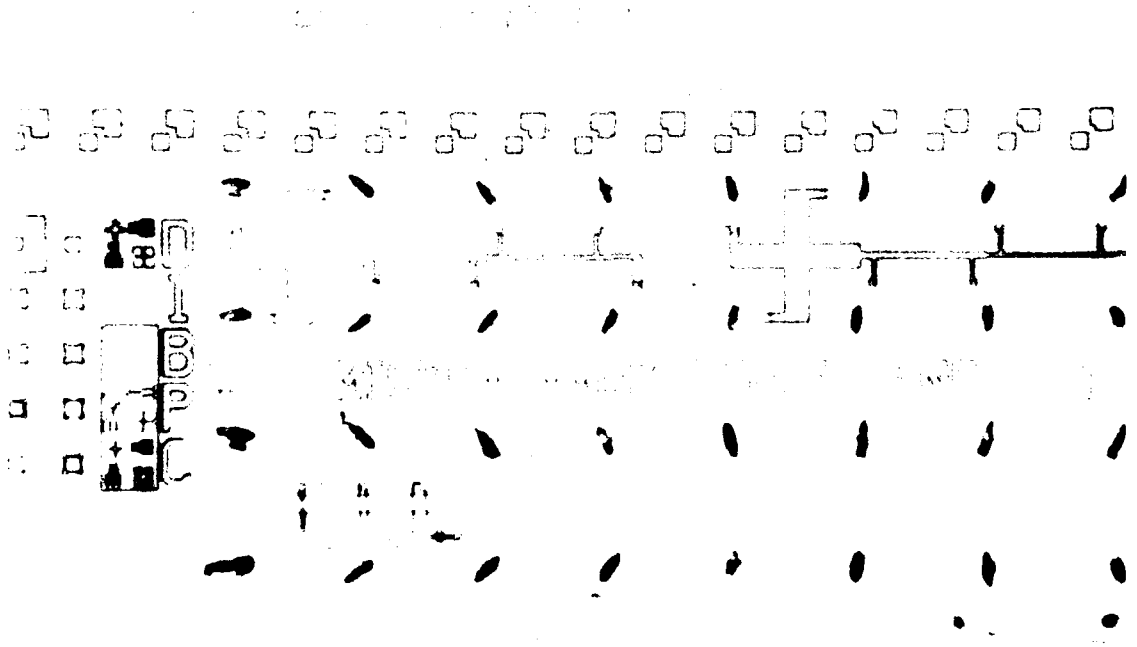


Figure 4-9: MOSIS standard test strip, left side

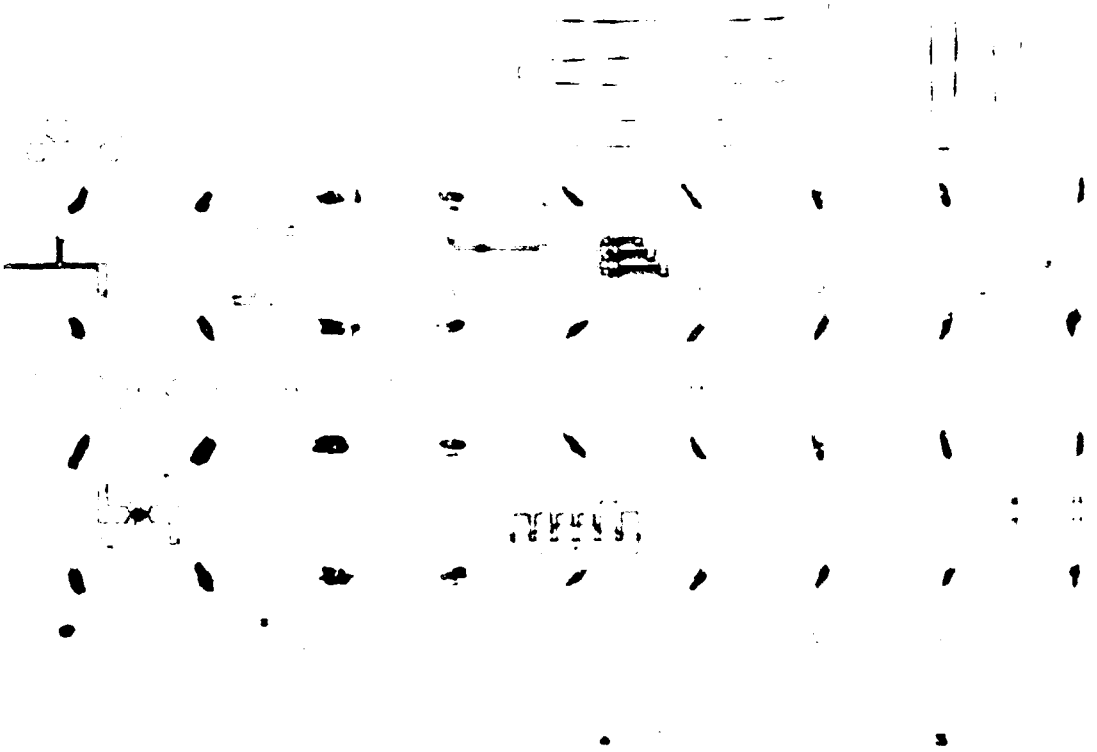


Figure 4-10: MOSIS standard test strip, middle

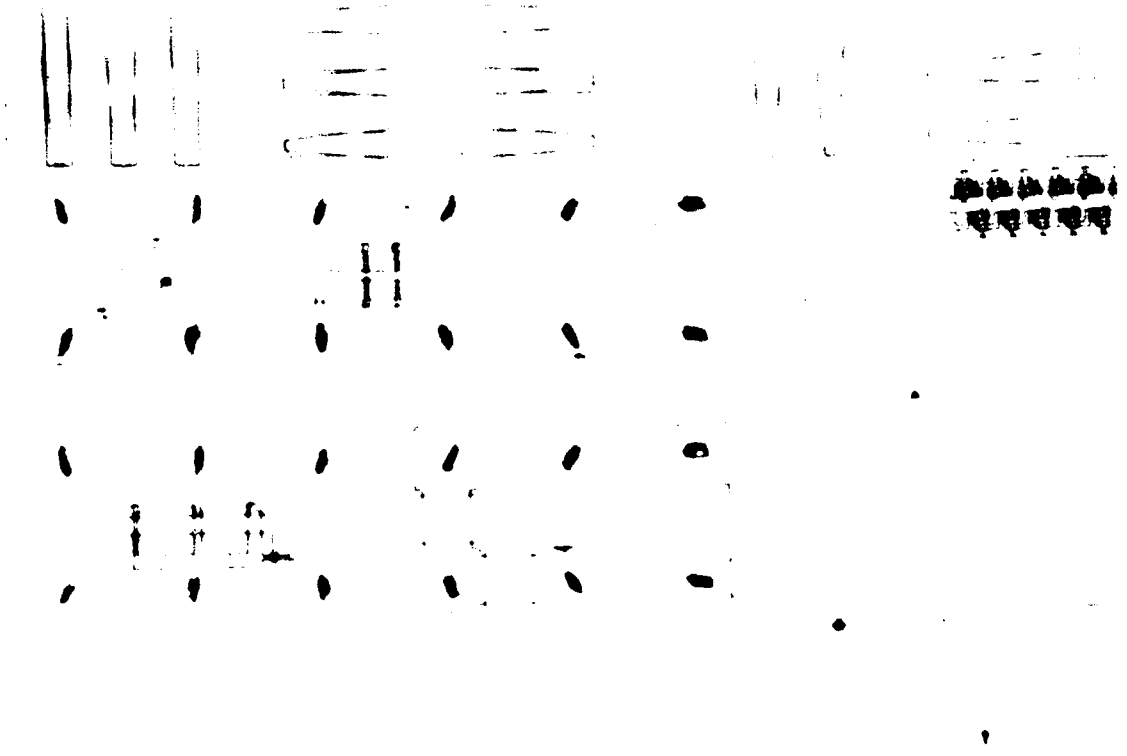


Figure 4-11: MOSIS standard test strip, right side

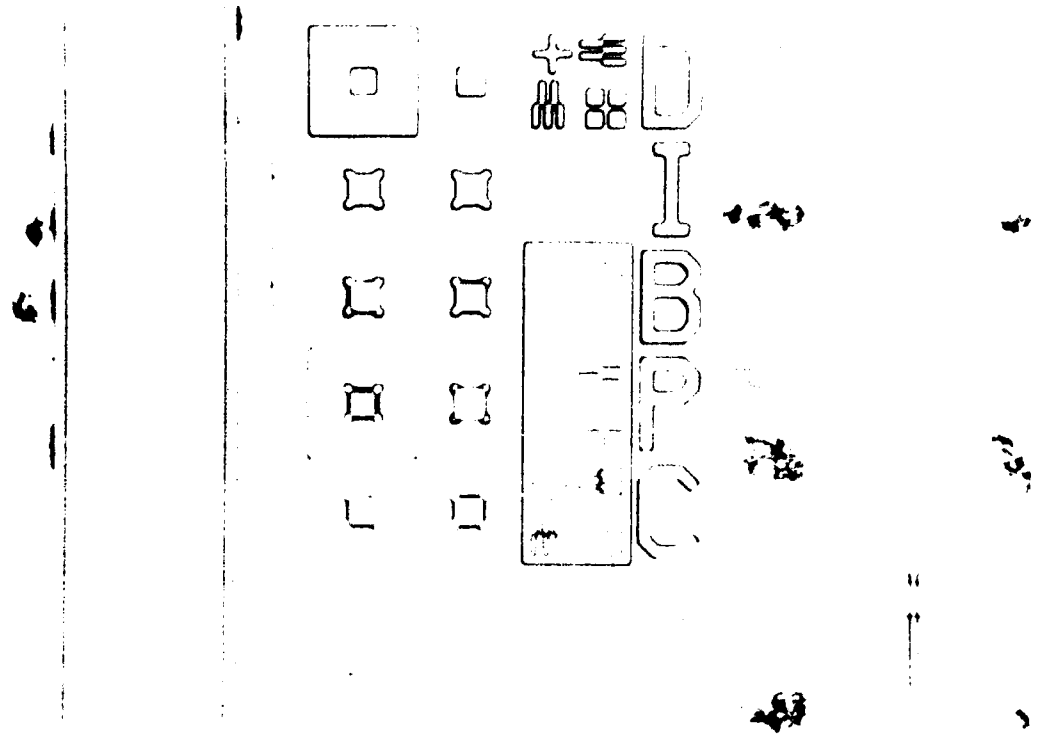


Figure 4-12: MOSIS critical dimensions and alignment marks

Figure 4-13: MOSIS CKSIZE device

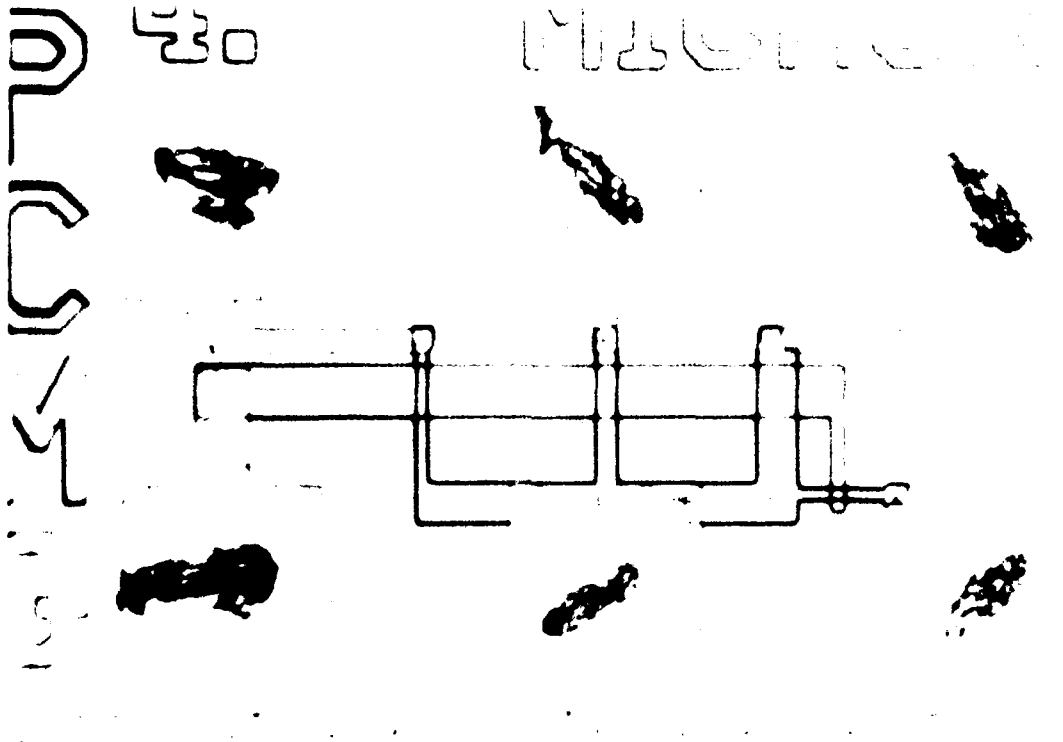


Figure 4-14: Variable width transistors

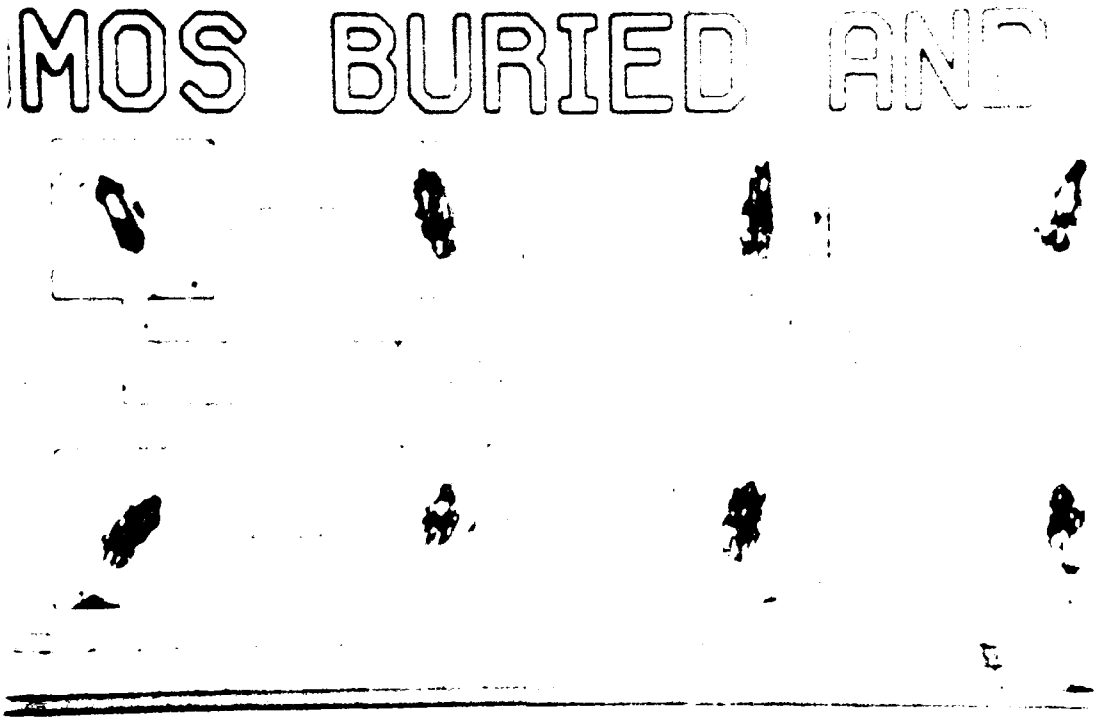


Figure 4-15: Sheet resistence and line width test device (metal)

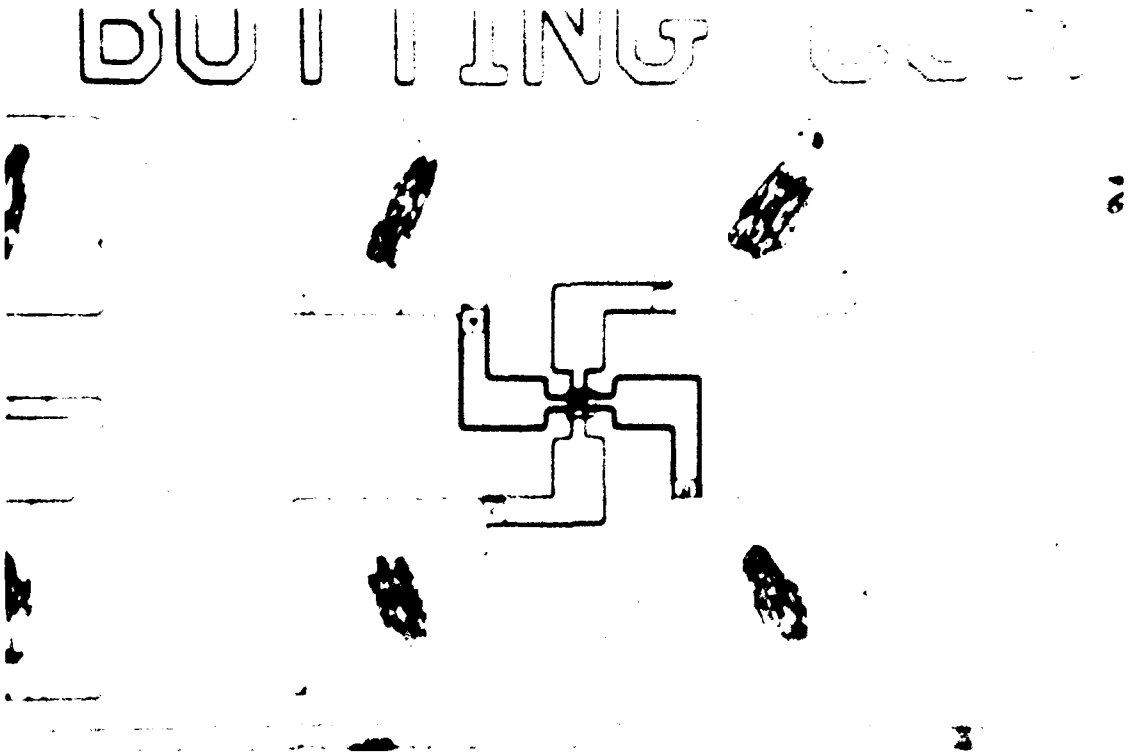


Figure 4-16: Buried contact test device

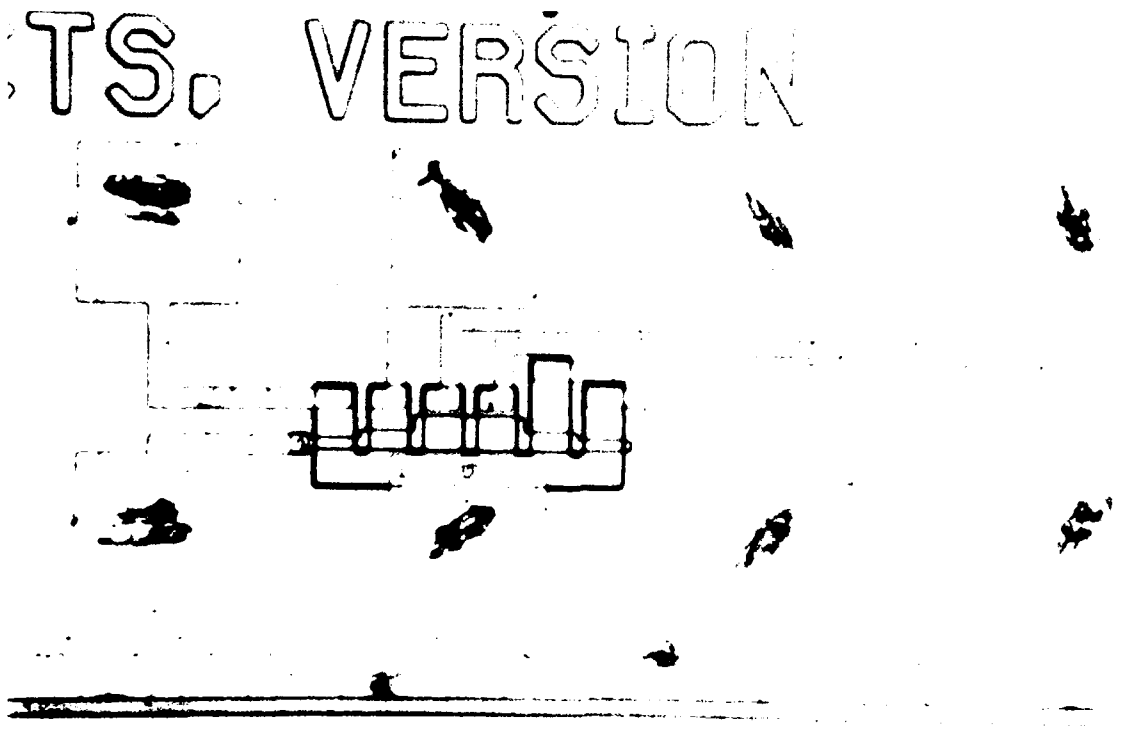


Figure 4-17: Variable length transistors

- Sheet resistance and line width measurement test structures (Fig. 4-15).
- Contact resistance measurements (Fig. 4-16).
- Parasitic field oxide transistor threshold measurements.
- Gate oxide capacitance and oxide thickness measurements.
- Simple inverters for measurement of inverter transfer function.
- Ring oscillator for evaluation of basic circuit speed.

Reduction of data from the test strip results in 91 parameters for each wafer. These parameters are summarized over the wafer lot and distributed to the community along with the packaged devices. The test strip data is also used as the primary input to SPICE Parameter reduction programs.

Test Strip

The test strip currently in use contains a few basic transistors and interconnects that allow measurement of some basic parameters at each die location on each wafer from a fabrication run. The test strip measurements are taken by automatic wafer-probing and provide information about transistor threshold voltage, sheet resistance of various layers, inverter gain (for inverters of various ratios), contact resistance, leakage current, and inverter threshold.

In addition to the basic transistor measurements, a few basic building blocks are used to ascertain circuit performance. Examples of these building blocks include: an inverter; an inverter fed with a minimum size pass transistor; long conductors for direct measurement of timing delays; and a few stages of a dynamic shift register to measure dynamic node storage times.

Figures 4-9, 4-10, and 4-11 show the standard MOSIS test-strip (Version 12, here). On the left of Fig. 4-9 are the standard MOSIS critical-dimension and dual-polarity alignment marks. Figures 4-14, 4-15, 4-16, and 4-17 show in more detail some of the test devices which belong to this test-strip.

Yield Monitors - "Drop-ins"

The test dies ("drop-ins") are intended to provide users with information about yield for each fabrication run. Measurements of structures on this die provide a designer with information about the probability of his having a defectless die. One of the structures we have been testing is a large (approximately 4K bits) random access memory based on lambda minimum spacing design rules, which provides a direct measure of fatal defects per unit area.

In addition, many Random Faults Structures devices are used in order to determine yield probability. Experience shows that the yield model used by MOSIS produces reasonable results. Other methods for estimating fabrication yield are being explored.

4.3.6 Data Analysis

Data collected from the measurement of test strips and drop-ins are processed to gather statistical information about the fabrication run. These processed data are printed out in various formats to show test results by wafer ID and by die location on each wafer. In addition to the publication of measurement results, the derived SPICE parameters are also made available to the designers. The MOSIS crew uses these summarized results in the following ways:

1. Wafer acceptance: The measurement results are used to determine if the fabricator has met required transistor thresholds, inverter thresholds, inverter gain, etc., for each wafer lot.
2. Wafer selection: Each fabrication run yields a few more wafers than are required to satisfy the needs of the users. Wafers to be cut, packaged, wire bonded, and shipped to the users are selected by using measurement results to find the "best of the lot."
3. SPICE parameters: Basic parameters for use in SPICE simulation of circuits (implemented with the design rules supported by MOSIS) are extracted from test strip measurements.

The measurement data are always made available to the fabricators and on several occasions were used for the fine tuning of the fabrication line.

Probe station operational

The probe station has been operational at ISI since January 1982 and now replaces the probe service that was so graciously provided by MIT Lincoln Laboratory since the MOSIS service began. The new probe facility is much faster and more capable than the probe station used by Lincoln Laboratory. This greater speed is used to provide a greater range of tests to be included on the test strip and on the "drop-in" and still be able to complete a probing session within a working day. The increased range of tests is used to give the user community a more detailed view of fabrication parameters to help them design additional chips.

Figure 4-8 shows the probe station operating on a standard set of 2-by-10 probing pads in the standard MOSIS test strip.

Fabrication runs completed between July 1981 and June 1982

Of the fabrication runs listed below, runs ZsaZsa and Delia are especially notable. The ZsaZsa run contains two oversized projects: RISC (from UC Berkeley) and GE-Geometry Engine (from Stanford University). The Delia run is a good example of a multiple project wafer. They are shown in Figures 4-18 and 4-19, respectively.

Run ID	Run name	Masks	Wafer	Feature size
M17M:	------	MicroMask	AMI	4um
M18N:	------	UltraTech	ZyMos	5um
M19P:	------	UltraTech	ZyMos	5um
M19Q:	------	UltraTech	ZyMos	5um
M19R:	------	UltraTech	ZyMos	5um
M1OS:	------	UltraTech	ZyMos	5um
M1DT:	Tanya	UltraTech	HP	4um
M1DU:	Ursula	UltraTech	HP	3um
M1DV:	Victor	UltraTech	ZyMos	5um
M1DX:	XMAS	UltraTech	ComDial	5um
M21Y:	Yolanda	UltraTech	ZyMos	5um
M21Z:	ZsaZsa	UltraTech	AMI	4um

M22B:	Barbra	UltraTech	ZyMos	5um
M23C:	Chloe	UltraTech	ZyMos	5um
M24D:	Delia	UltraTech	ZyMos	5um
M24E:	Eva	UltraTech	ComDial	5um
M25H:	Honey	Sierracin	HP	4um
M25K:	Keith	Sierracin	ZyMos	5um
M26L:	Lee	Sierracin	ZyMos	5um

4.3.7 The VLSI Design Library

The MOSIS CIF-design library offers some basic circuits such as bonding pads, shift register buffers, super-buffers, PLA fragments, and arithmetic circuits. These circuits were either designed by the MOSIS crew or contributed to the MOSIS library by designers at various sites, including Xerox PARC, Stanford University, UC Berkeley and Caltech.

The library has entries for nMOS, CMOS/bulk and CMOS/SOS in various feature sizes.

Several test devices are also included in the library.

Designers may retrieve information from the MOSIS library in a fully automated way, over either of the electronic mail systems.

4.4 FUTURE WORK

During the next reporting period the VLSI Project will extend the MOSIS service to include the following:

1. Smaller feature sizes.

   - The current nMOS design rules can be supported at a feature size of 5 to 3 microns. It is expected that this will be reduced even further.
   - Currently, MOSIS supports CMOS/bulk in 5 and 3 microns. We expect to move very soon (probably before the publication of this report) to a 1.25 micron feature size.
   - MOSIS also supports CMOS/SOS in 4 micron; we expect to reduce this size within a year.

2. With the increase in the number of technologies supported by MOSIS, it is necessary to increase the degree of automation in the wafer probing and data reduction process. In addition, it is necessary from time to time to change the design of parametric test structures to enhance the quality of data provided to the community. To accomplish this in a uniform manner a "generic MOS" test strip generation program will be developed which will guarantee that the test strips of all technologies reflect the changes identically. In order to allow more rapid evaluation of yield monitors (which are currently tested as packaged devices) a functional test capability at wafer probe will be added to the wafer probe station.
3. Improved and expanded testing capabilities. Wafer probe testing is performed upon every wafer fabricated for the user community. A standard test structure on each project

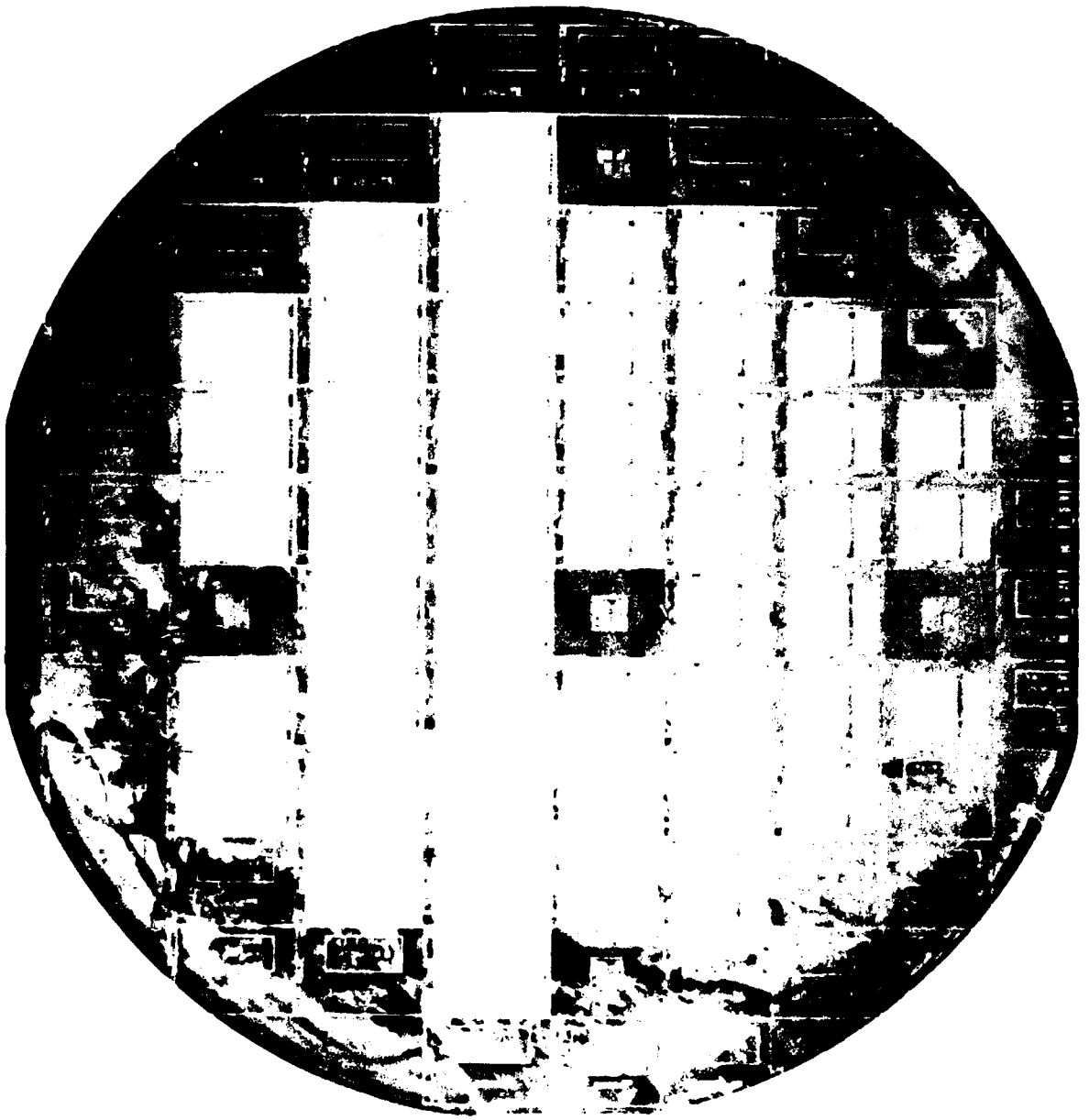


Figure 4-18: ZsaZsa--M21Z

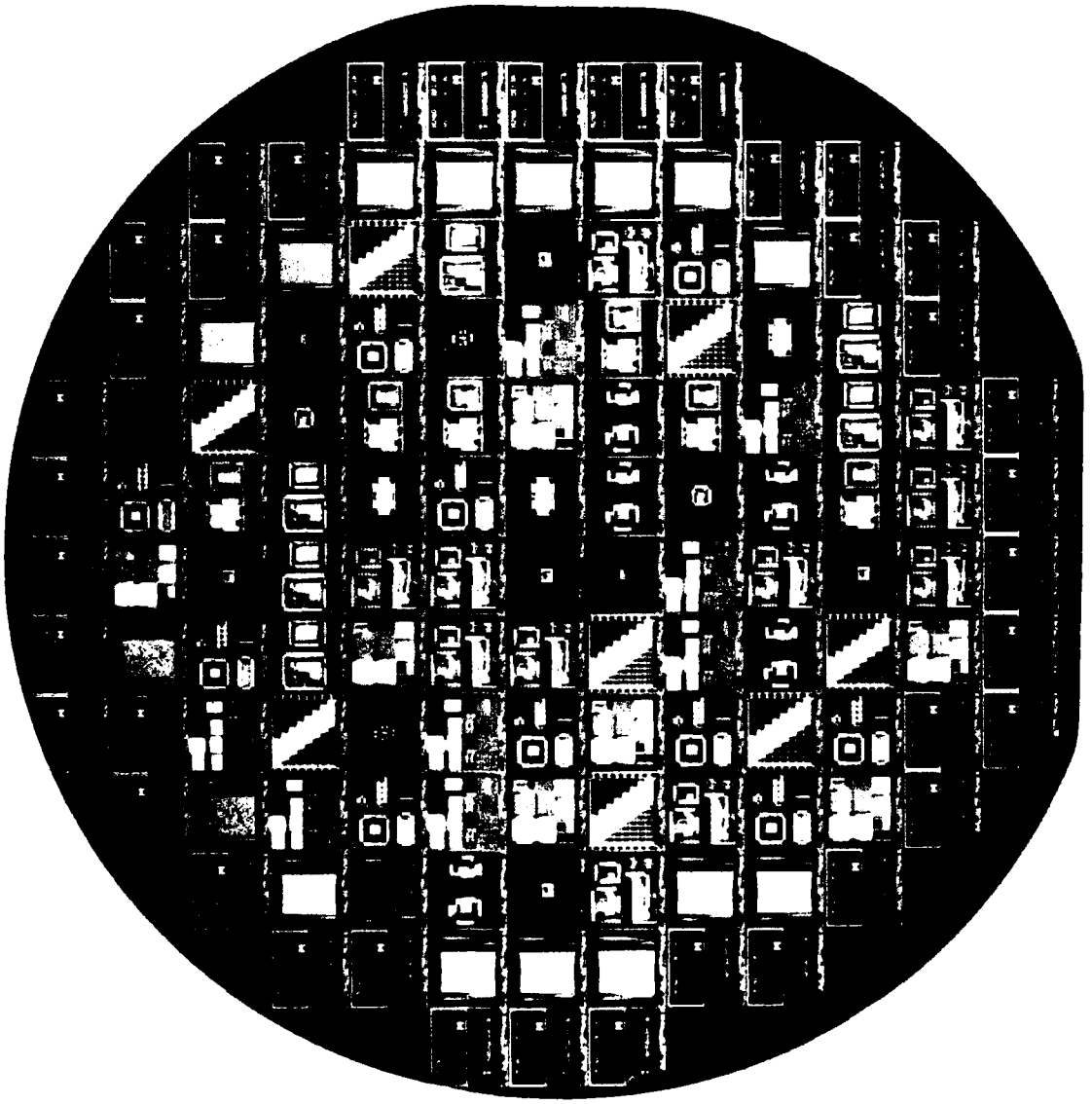


Figure 4-19: Delia--M24D

die is used to measure the quality of the fabrication. This testing is being improved and expanded to provide information on average yield for typical fabrication runs and to extract and distribute transistor model parameters to the users. An effort is under way to develop yield measuring test structures that will accurately model the expected yield for typical user designs. In addition, the current test strip and drop-in test die are being redesigned to simplify testing and improve the accuracy of parameter extraction.

REFERENCE

1. Foderaro, J. K., K. S. Van Dyke, and D. A. Patterson, "Running RISCs," *VLSI Design*, September/October 1982, 27-32.

5. INTERNETWORK CONCEPTS RESEARCH

Research Staff:
Jon Postel
Danny Cohen
Carl Sunshine
Paul Mockapetris
Greg Finn

Research Assistants:
Alan Katz
Dave Smallberg

Support Staff:
Ann Westine

5.1 PROBLEM BEING SOLVED

This project explores the design and analysis of computer-to-computer communication protocols in multi-network systems. The project has had three task areas: 1) Analysis, 2) Applications, and 3) Design and Concepts. Protocol Analysis is concerned with the correctness of protocols, in particular the Transmission Control Protocol (TCP). TCP is a connection-style, host-level protocol that provides reliable, ordered delivery of streams of data. Protocol Applications is concerned with the development of demonstration internetwork applications, in particular a prototype computer message system. Protocol Design and Concepts is concerned with the development of network and transport protocols, in particular the Internet Protocol (IP) and TCP, and seeks new approaches in the application of packet switching to communication problems. IP is a datagram-based, gateway-level protocol that provides the addressing, routing, and fragmentation/reassembly functions in the internetwork.

5.2 GOALS AND APPROACH

The long term goals of this research are to provide appropriate and effective designs for the primary user service applications in the internetwork communication environment. The designs are based on a set of host and gateway level protocols that provide the full range of service characteristics appropriate to a wide range of applications and that have been specified and analyzed to ensure their correct operation.

Our approach has been to pursue in parallel the analysis, application, and design of protocols. The interaction of these activities provides valuable insights into problems and potential solutions.

We have identified several program and protocol analysis tools and techniques that show promise of being of assistance in the study of protocols. We have explored the value of these tools and techniques by applying them to a series of example protocols. The example protocols incorporate features of the TCP.

We have selected computer mail as the application to use as a focus for demonstrating internetwork service. Within this application area we are developing two distinct mail systems: (1) a Text Mail system, and (2) a Multimedia Mail system. The Text Mail system has evolved from the long existing ARPANET mail system. Our contribution to this multi-organization activity focuses on the design and initial implementation of the TCP-based Simple Mail Transfer Protocol (SMTP). The Multimedia Mail system is an experiment to learn about the future direction of computer mail systems.

The goal is to provide mechanisms for computer-oriented data structures to communicate various types of data in messages, including text, graphics and voice. In both mail systems, our interest is the communication mechanism, not the user interface.

We have assisted in the design of several protocols in the internet family. The areas of addressing, routing, and multiplexing are particularly subtle and require careful attention. We have focused attention on these areas and have explored many options in technical discussions and memos. Our approach is to develop an understanding of these technical issues and to advocate the inclusion of general-purpose supporting mechanisms in the protocol specifications.

5.3 SCIENTIFIC PROGRESS

5.3.1 Protocol Analysis

Our work this year has been divided into three major areas: survey, in-depth studies, and standards activities.

In the survey area, our major activity has been organization of an international workshop on protocol specification, testing, and verification attended by over 40 experts from 13 countries. Contributions from most of the major centers of research in this area were solicited, and the resulting proceedings [15-17] represents a wide sample of current state of the art in this area. We have also continued distribution of the results of our own survey work in the literature [7,10].

Last year we experimented extensively with the Affirm system. Our in-depth experiments this year have focused on three other systems, each with a particular promising orientation. The Ina Jo [registered trademark of the System Development Corporation] system explicitly supports abstract machine models. The Gypsy system emphasizes external behavior specifications and prohibits internal state specifcations of processes. The State Delta system features an automatic symbolic execution prover rather than requiring interaction with the user.

Our experiments show that each of these features has merit, but also causes difficulties in some cases. While Affirm seems the most polished, convenient, and effective system in general, it, too, has major shortcomings. No single system currently available provides all necessary features, but each is quite effective in certain areas. This work is documented in [24,32].

We have also continued to document and distribute the results of our work with Affirm [20,21].

Our third task area has involved participation in standards activities in order to promote wider use of the more rigorous specification methods we and others have been developing. This work has included development of specification standards within ISO SC 16 (for use with Open Systems protocols and services), collaboration with the System Development Corporation in the development of new IP, TCP, and Telnet protocol specifications [27,28,29], and comments on the transport protocol being developed for the National Bureau of Standards by Bolt Beranek and Newman [30,31].

5.3.2 Protocol Applications

The Multimedia Mail system

This message system model takes the view that the message service can be divided into two activities: message reading and composition, and message delivery. The message reading and composition is an interactive activity in conjunction with a User Interface Process (UIP). The message delivery activity may be carried out by background processes called Message Processing Modules (MPMs). Our work concentrates on the message delivery aspects and leaves the development of sophisticated user interfaces to other projects (e.g., Cooperative Interactive Systems--see Chapter 1).

The internetwork multimedia message system is concerned with the delivery of messages between MPMs throughout an interconnected system of networks. It is assumed that many types of UIPs will be used to compose and display messages delivered by MPM processes. The MPMs exchange messages by establishing full duplex communication and sending the messages in a tightly specified format. The MPMs may also communicate control information by means of commands.

A message is formed by a user interacting with a UIP. The user may utilize several commands to create various fields of the message and may invoke an editor program to correct or format some or all of the message. Once the user is satisfied with the message, it is "sent" by placing it in a data structure shared with the MPM.

The MPM takes the data, adds control information to it, and transmits it. The destination may be a mailbox on the same host, a mailbox on another host in the same network, or a mailbox in another network.

The MPM calls on a reliable communication procedure to communicate with other MPMs. In most cases, this is a Transport Level protocol such as the TCP. The interface to such a procedure typically provides calls to open and close connections, and to send and receive data on a connection.

The MPM receives input and produces output through data structures that are produced and consumed by UIPs or other programs.

The MPM transmits the message including the control information in a highly structured format using typed data elements in a machine-oriented yet machine-independent data language.

After the second specification of the MPM was completed, an implementation was begun. That implementation is now largely complete. The MPM is now available for testing at TOPS-20 installations that wish to experiment with multimedia mail. Maintenance and extensions are provided for the MPM as users report problems or suggestions. Several sites are now using the MPM.

A simple UIP has been developed for testing and demonstrating the multimedia mail system.

The Text Mail system

The goal of this mail system is to provide improvements to the existing mail procedures and to bridge the gap between the different interprocess communication systems presented by the Network Control Protocol (NCP) and TCP host-to-host protocols, providing unified mail service for existing mail users on the ARPANET and other networks on the Internet. This service interfaces NCP-only, TCP-only and other hosts via a system of SMTP relay processes in the hosts.

During the past year, the prototype MTP system has been largely supplanted by SMTP. We have implemented an initial SMTP server for TOPS-20 and have placed it in service.

5.3.3 Protocol Design and Concepts

A significant accomplishment was the development of the NCP/TCP Transition Plan [5], which will be complete by January 1983. We also revised the specifications for the IP, TCP, and ICMP [11-13]. The Internet Protocol is a datagram-based, gateway-level protocol that provides the addressing, routing, and fragmentation/reassembly functions in the internetwork. We discussed service mappings and address mappings separately in [18,19]. The Transmission Control Protocol is a connection-style, host-level protocol that provides reliable ordered delivery of streams of data. We discussed pre-emption separately in [14]. The Internet Control Message Protocol is an integral part of the IP that provides for error and control messages between destination hosts or gateways and source hosts. All the documents relating to these specifications were compiled as part of the Internet Protocol Transition Workbook [33], available from SRI International.

In addition, we have produced a new specification for the SMTP [4,8], and we have contributed to general planning of the internet communication system, especially the hierarchical domain-naming convention [9], and a variety of issues in mail transfer [1-3] and communications in general [26].

We have continued to support the Internet Working Group through presentations, meetings, and preparation of meeting notes, agendas, and other routine documents [6,22,23,25]. We also coordinate the monthly report for the ARPA Internet Program.

5.4 IMPACT

5.4.1 Protocol Analysis

The use of more precise specification methods will facilitate cheaper, faster, and more reliable implementation of the ever-increasing number of communication protocols in DoD computer networks. The research described here has already had some impact on major protocol development projects sponsored by the Defense Communications Agency (DCA) and by national and international standards groups. The previously widespread informal narrative specification methods are being augmented by more formal specifications, particularly of state transition or abstract machine variety.

The development of protocol verification techniques also promises to improve reliability and reduce debugging time in implementing network systems. By allowing analysis of protocol designs prior to actual implementation, these techniques can help detect problems earlier. Esoteric bugs that would probably escape detection by ordinary testing and debugging may also be revealed by formal verification. Several results of this nature have already been discovered and applied to the evolution of TCP.

This research program has also had an impact by influencing other research projects, particularly in the area of program verification, toward developing analysis techniques that are applicable to computer networks. The results of our work have been widely reported in conferences, journals and books, and are often cited by others in the field.

5.4.2 Protocol Applications

Computer mail is the most significant use of the new communication capability provided by packet switching networks. Our work to extend the range and capabilities of computer mail will have important consequences for DoD.

The potential for multimedia communication in a computer-assisted environment is great. The ability to communicate diagrams or maps and then to talk about them will increase the effectiveness of remote communication tremendously. The combination of text, speech, graphics, and facsimile into a common framework and data structure may have substantial impact on other applications as well.

The power of a communication system is directly related to the number of potential communicants. For computer mail, this means that the power of a system is related to the number of people who have access to that system. To have access to a computer mail system requires the use of compatible components: terminals, programs, and protocols. Our work on protocols and programs will increase the power of computer mail by enlarging the set of compatible components.

5.4.3 Protocol Design and Concepts

The selection of the IP and TCP protocols by the DoD as the basis for a DoD internetwork protocol standard shows the impact of the work of the ARPA community on DoD communication systems. The development of the Defense Data Network (DDN) by the Defense Communication Agency (DCA) will be a major use of these protocols in an operational military system. This influence is demonstrated further by the mounting requests for information about IP/TCP from companies interested in producing commercial products or bidding on government contracts.

Through our participation in discussions at the Internet Working Group meetings, and in technical meeting with other contractors, we have successfully influenced the development of many protocols and protocol features.

5.5 FUTURE WORK

5.5.1 Protocol Analysis

We have concluded our work in protocol verification and have reported on the experience with the verification systems used and the results of attempting to verify TCP. We are now dropping this task area.

5.5.2 Protocol Applications

We will continue support for the operation of the MPMs at several TOPS-20 sites, support and development of the simple UIP for experimentation, as well as design and specification of protocols and formats for multimedia mail.

We will continue to assist with the integration of SMTP with other Internet mail components and

examine issues relating to the specification of mail protocols, mail addressing systems and other aspects of mail in the Internet environment.

We will design and develop a Text Mail/Multimedia Mail translation interface. This will involve an interface between the MPM and SMTP programs.

In addition, we will investigate performance issues and application dependencies of different underlying network technologies, using FTP to study performance trade-offs using Ethernet, WBNet and ARPANET.

5.5.3 Protocol Design and Concepts

We will continue our contributions to the development of internet gateway level (IP) and host level (TCP) protocols. In particular, we will focus on gateway protocol issues, protocol architecture, and new protocol designs. New protocol design areas include Host Name Servers, Host Who Are You Servers, File Access Protocol, and a Bitmap Telnet Protocol.

We will continue to raise important issues for discussion in the Internet Working Group and to seek out and develop new opportunities to utilize the capabilities of packet-switched communication systems.

REFERENCES

1. Sluizer, S., and J. Postel, "Mail Transfer Protocol: ISI TOPS20 Implementation," USC/Information Sciences Institute, RFC 784, July 1981.

2. Sluizer, S., and J. Postel, "Mail Transfer Protocol: ISI TOPS20 File Definitions," USC/Information Sciences Institute, RFC 785, July 1981.

3. Sluizer, S., and J. Postel, "Mail Transfer Protocol: ISI TOPS20 MTP-NIMAIL Interface," USC/Information Sciences Institute, RFC 786, July 1981.

4. Postel, J., "Simple Mail Transfer Protocol," USC/Information Sciences Institute, RFC 788, November 1981.

5. Postel, J., "The NCP/TCP Transition Plan," USC/Information Sciences Institute, RFC 801, November 1981.

6. Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC 790, September 1981.

7. Sunshine, C., "Formal protocol specification," in C. Solomonides (ed.), *State of the Art Report on Network Architectures*, Pergamon Infotech, 1982.

8. Postel, J., "Simple Mail Transfer Protocol," USC/Information Sciences Institute, RFC 821, August 1982.

9. Su, Z., and J. Postel, "The Domain Naming Convention for Internet User Applications," Network Information Center, SRI International, RFC 819, August 1982.

10. Sunshine, C., "Formal modeling of communication protocols," in S. Schoemaker (ed.), *Computer Networks and Simulation II*, North-Holland, September 1982.

11. Postel, J., "Internet Protocol - DARPA Internet Program Protocol Specification," USC/Information Sciences Institute, RFC 791, September 1981.

12. Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," USC/Information Sciences Institute, RFC 792, September 1981.

13. Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification," USC/Information Sciences Institute, RFC 793, September 1981.

14. Postel, J., "Pre-emption," USC/Information Sciences Institute, RFC 794, September 1981.

15. Sunshine, C. (ed.), *Proceedings of the Second International Workshop on Protocol Specification, Testing, and Verification*, North-Holland, May 1982.

16. Sunshine, C. (ed.), *Computer Networks*, Special Issue on Protocol Specification, Testing, and Verification, to appear in early 1983.

17. Sunshine, C. (ed.), *IEEE Transactions on Communications*, Special Issue on Protocol Specification, Testing, and Verification, December 1982.

18. Postel, J., "Service Mappings," USC/Information Sciences Institute, RFC 795, September 1981.

19. Postel, J., "Address Mappings," USC/Information Sciences Institute, RFC 796, September 1981.

20. Berthomieu, B., *Algebraic Specification of Communication Protocols*, USC/Information Sciences Institute, RR-81-98, December 1981.

21. Schwabe, D., "Formal specification and verification of a connection-establishment protocol," in *Proceedings of the Seventh Data Communications Symposium*, Mexico City, IEEE, October 1981.

22. Postel, J., "Computer Mail Meeting Notes," USC/Information Sciences Institute, RFC 805, February 1982.

23. Postel, J., "Multimedia Mail Meeting Notes," USC/Information Sciences Institute, RFC 807, February 1982.

24. Sunshine, C., "Experience with four automated verification systems," in C. Sunshine (ed.), *Proceedings of the Second International Workshop on Protocol Specification, Testing, and Verification*, North-Holland, May 1982.

25. Postel, J., "Summary of a Computer Mail Service Meeting Held at BBN on 10 January 1979," USC/Information Sciences Institute, RFC 808, March 1982.

26. Sunshine, C., Danny Cohen, and J. Postel, "Comments on Rosen's Memos (IENs 184, 187-189)," USC/Information Sciences Institute, IEN 191, July 1981.

27. Simon, G. A., *DCEC Protocols Standardization Program/Protocol Specification Report*, System Development Corporation, TM-7038/204/00, July 1981.

28. Bernstein, M., *DCEC Protocols Standardization Program: Proposed DoD Internet Protocol Standard*, System Development Corporation, TM-7038/205/01, December 1981.

29. Bernstein, M., *DCEC Protocols Standardization Program: Proposed DoD Transmission Control Protocol Standard*, System Development Corporation, TM-7038/207/01, December 1981.

30. Sunshine, C., "Comments on the NBS Transport Protocol Proposal," USC/Information Sciences Institute, IEN 195, August 1981.

31. Sunshine, C., "Comments on 'Formal Service Specification of the Transport Protocol Services' (April 1982 draft by Bolt Beranek and Newman for the National Bureau of Standards)," USC/Information Sciences Institute memo, June 1982.

32. Sunshine, C., "Protocol Specification and Verification Work at USC/ISI: Summary Report," USC/Information Sciences Institute, IEN 211, August 1982.

33. Feinler, E., *Internet Protocol Transition Workbook*, Network Information Center, SRI International, March 1982.

6. COMMAND GRAPHICS

Research Staff:
Richard Bisbey II
Benjamin Britt
Pamela Finkel
Dennis Hollingworth

Consultant:
Danny Cohen

Support Staff:
Lisa Holt

6.1 PROBLEM BEING SOLVED

As C2-related information is increasingly maintained in computer-based form, computers need to take more active roles in presenting that data. Computer-generated graphics must supplant volumes of batch-generated printer listings to allow the military to deal with the large quantities of information available and make more timely decisions. For example, online computer decision aids will be needed to calculate and graphically display the potential outcomes of alternative strategies. Such a capability must be available in normal, as well as crisis, mode, with particular attention paid to mobility and survivability in crisis mode.

To meet future military needs, the next generation of command and control graphics systems will need to have the following attributes:

- They must be adaptable to the computation and communications resources that are available. The unpredictability of the communications bandwidth available and the location of and accessibility to computational resources during a crisis situation, together with the need to optimize use of available resources to meet the situation, require that the graphics system be tailorable to a wide variety of processor/communications combinations.
- They must be flexible to adapt to and utilize different types of display devices. A wide variety of display devices will be available, ranging from large screen color displays in command centers to more modest hand-held displays in the field. A graphics system must accommodate all of these device types and be capable of displaying the graphics output of any available application program.
- They must be usable in a transnetwork environment. Command centers, host computers, and users will be interconnected by digital networks that will include terrestrial, radio, and satellite communications. Graphics systems for command and control must be designed to operate in this environment.
- They must support the creation of pictures for use outside the immediate application-program environment. Graphics systems for command and control must provide the capability for storing an application-program-generate graphics picture in a "graphics file," transmitting the file over a network, and incorporating that picture in the graphic output of another application program, all in a display-device-independent fashion.

More recently, additional requirements have been identified. In particular, evolution of the notion of command mobility manifested by the HERT (Headquarters Emergency Relocation Team) concept has resulted in the identification of two additional requirements:

- Part or all of the graphics system must be portable to allow for a relatively high bandwidth of graphic information exchange between the application program and an air, sea, or land mobile user.

- Neither the graphics system hardware nor the support processing hardware must be prohibitively expensive in large quantities as is the case with the large computer mainframes on which the distributable graphics system was first implemented. Budgetary constraints imposed on equipment and military hardware must be anticipated and taken into consideration in terms of the cost of graphics system hardware.

A second area being addressed relates to graphic communications between distributed participants. Recent recognition of the need for improved conferencing capability to counter costs and difficulties associated with assembling a particular set of individuals at the same time has resulted in experiments in video teleconferencing. The conference participants assemble at designated conference centers around the country and a live video image is transmitted between the various centers.

While this approach allows individuals to have a conference much as if they were in the same room, it does not exploit computer technology to make computer graphic information available to the conference participants, i.e., there are video and audio links between the participants, but not a computer graphic link interconnecting the participants with a computer. Moreover, the high bandwidth required for such interaction makes it quite expensive and requires substantial planning and preparation to initially establish the conference capability. Furthermore, the approach is not conducive to rapid reaction to a crisis situation where the participants' responsibilities and circumstances may not even allow them to assemble at a conferencing center. The actual locations of various respondents may not be predictable ahead of time. In fact, the respondents may be widely dispersed, so it must be possible to establish voice and data links relatively close to the actual location of individual respondents with minimal effort.

6.2 APPROACH

ISI has developed a set of generic graphics primitives (Graphics Language) by which pictures can be described and interacted with at the application program level [1]. The particular graphics model used for the language was based on structuring pictures as sets of subpictures which are absolute-transformed segments, as defined by Newman and Sproull [3]. The graphics primitives are transformed at program execution time into specific operations and display modes appropriate to the device to which the system is connected.

ISI also defined an architecture for distributable graphics [2]. The architecture allows a graphics system to be constructed as a series of isolatable functions that are pairwise connected by any available intraprocess/interprocessor communications mechanism including telephone, radio, or digital network/internetwork links. Information is communicated between functions using a uniform protocol. The architecture permits a wide variety of distributions ranging from clustering all functions on a single host to distributing each to a different host. Included within the architecture are functional modules that allow the creation of sequential files containing display-device-independent graphics and the incorporation of such files into an existing application's graphics. The distributed architecture has been implemented for a number of different display device types as well as several example applications including Situation Display, demonstrating the viability of a distributable graphics system architecture and the utility of such a system.

The perceived need for a graphics communications interface resulted in development of a capability for sending graphic pictures over the network. Based on experiments using a graphics file display utility program, ISI began defining a collaborative graphics capability to allow users to

graphically interact with one another across a computer network in various ways with minimal effort. Interconnecting users with a computer graphics link has major benefits over full video. It allows participation by users with small, low cost graphic terminals interconnected by a modest bandwidth digital link. Furthermore, the users need not be assembled in centralized conference centers, but can be distributed in possibly remote or portable locations. As a first effort in this area, ISI developed a Briefing Aid application.

6.3 PROGRESS

To date, the Graphics System has been implemented on PDP-10s and 20s operating the TENEX and TOPS-20 operating systems, and PDP-11/70 remote site modules operating the UNIX operating system, all of which are large and expensive mainframe computers. To meet the requirements for portability and low cost, ISI is reimplementing the Backend portion of the Graphics System on a Motorola MC68000 microprocessor. Unlike a PDP-10 or PDP-11 computer that occupies one or more equipment racks, a microprocessor-based system may occupy one or two 16x9 inch cards, small enough to be packaged with the display device. This allows portability of operation while retaining maximum graphics display performance. Second, a microprocessor-based system can provide the same functional capability of a mainframe host at a fraction of the cost, resulting in a substantial cost savings. In the process of reimplementing the Backend for the MC68000, a VAX version is also being created.

This period also saw the completion of the Briefing Aid Application. Briefing Aid permits the creation and presentation of briefings using computer graphics rather than conventional film media. The online graphics output of any command and control application program, such as Situation Display, can be readily incorporated into a briefing. Briefing slides can be displayed sequentially under the control of a separately prepared briefing script or displayed individually under direct briefer control. The distributability and display-device-independent properties of the graphics system permits briefings to be given anywhere a display terminal can be located, such as in a commander's office or at a mobile headquarters. Briefing Aid also includes a speech capability to allow LPC vocoded speech information to be included in the briefing and played back through an LPC voice synthesis chip under control of the briefing script. In addition to the Briefing Aid application, two support programs were written. First, to aid in the creation of simple slides, a graphics editor was written. Second, to enable reediting previously generated graphic files, a graphics file to Graphics Language translation program was written.

6.4 IMPACT

The principal impact of this work will be felt in military environments where command mobility is paramount and a portable, network-based graphics capability is necessary for information presentation to facilitate command decisions. This work will make the Graphics System available in [illegible] and other mobile environments with major display performance improvement over that currently possible. The graphics system Backend will reside in the mobile environment itself, ensuring necessary communications bandwidth for good display performance. Minimal [illegible] and power requirements will result.

[illegible] system will be able to utilize ISI software for presenting computer-based briefings [illegible] graphical information exchange. This will allow increased flexibility in providing

decision information to individuals in command roles, allowing new approaches to command survivability through increased mobility.

6.5 FUTURE WORK

During the past year, ISI has been working on the design and implementation of a small, portable Graphics System Backend implemented on a Motorola MC68000 microprocessor. This Backend will allow graphic users to interact with an application on a remote host while retaining the graphic performance of a local host. It is now apparent that the same 16-bit microprocessor used to support the Backend functions is also powerful enough to support a Graphics System Frontend and a modest user application program. A major benefit is that it provides the graphics user with a complete, self-contained, highly portable graphics capability. When the user is connected to a network, both the application/Frontend and Backend can be either local or remote to the user. During those times when the user is disconnected from the network, the user retains the capability to run applications locally. ISI will produce a 68000-based Graphics System Frontend.

The Briefing Aid application described above allows an individual to utilize a computer to prepare and present a briefing to one or more recipients who are all located at the same facility. A much broader perspective of computer-assisted exchange of visual information can be adopted which encompasses not only briefings but also common blackboards and graphic conferencing. While each of these has dissimiliarities, each also has many attributes in common which suggests an integrated approach to providing such capability. We refer to this integrated approach as collaborative graphics and are directing resources to refining the concepts involved and producing a network based collaborative graphics capability. Such capability would allow for multi-site computer briefings, multiple individuals to connect to and view or update a common blackboard (status board) facility, and multi-site graphics conferences any of which might be augmented by a telephone connection for audio interaction.

REFERENCES

1. Bisbey, R., II, D. Hollingworth, and B. Britt, *Graphics Language*, USC/Information Sciences Institute, TM-80-18, 1980.

2. Bisbey, R., II, and D. Hollingworth, *A Distributable, Display-Device Independent Vector Graphics System for Command and Control*, USC/Information Sciences Institute, RR-80-87, 1980.

3. Newman, W. M., and R. F. Sproull, *Principles of Interactive Computer Graphics*, second edition, McGraw-Hill, 1979.

7. WIDEBAND COMMUNICATION

Research Staff:
Stephen Casner
William Brackenridge
Danny Cohen
E. Randolph Cole
Ian Merritt

Support Staff:
Lisa Holt
Jeff LaCoss
Robert Parker
Jerry Wills

7.1 INTRODUCTION

The Wideband Communication (WBC) project at ISI is one of several groups participating in the DARPA/DCA Wideband Packet Satellite Program. ISI's role is that of the experimenter, applying the Wideband Packet Satellite Network to new modes of communication, such as packet voice and packet video. This work is done in cooperation with projects at other sites.

The network itself is in the early stages of operation. Others in the program are responsible for the development of equipment and software for the satellite network nodes (ground stations and packet switches). Bringing the network to full operation and characterizing its performance is a cooperative effort involving the developers and experimenters alike.

7.2 PROBLEM BEING SOLVED

The military has supported packet switching technology since its inception with the ARPANET in the 1960's. Most of the ARPANET's use has been in research, but the transition from the ARPANET to the secure Defense Data Network (DDN) signals that the military is becoming a true user of packet switching, at least for data communication.

The first objective of the Wideband Program is to extend the technology for real-time packet communication, especially voice, so that it too can be put into practice. The DARPA Network Secure Communication (NSC) effort demonstrated in the mid-1970's that it is feasible to transmit real-time, secure speech through a packet network (the ARPANET). However, the effects of scale on a packet voice communications system remain to be investigated.

The NSC program used minicomputers with attached high-speed signal processors; such systems are flexible but relatively expensive, so only a few were used. Advances in VLSI (very large-scale integration) technology, combined with sufficient experience in packet speech protocols and software, have made possible the creation of a more economical voice terminal. Some 20 prototype voice terminal units have been produced by Lincoln Laboratory for use in the Wideband Program and other applications. Larger-scale production is possible at further reduced cost.

Likewise, the bandwidth of previously available long-haul packet networks has limited the number of simultaneous voice conversations to just a few. The wideband satellite channel has the capacity to simultaneously transmit about 20 uncompressed voice streams or a few hundred narrowband voice streams, although the current capabilities of the network packet switches limit this somewhat. The

Wideband Network is being used to test packet speech in an environment which more closely resembles future real-world applications, an environment in which the packet-switching technology can take advantage of the bursty nature of speech to demonstrate its efficiency in statistical multiplexing.

A study by the Network Analysis Corporation [4] showed that packet switching technology with an integrated voice/data network is the most economical way to meet DoD communication needs, as compared over a wide range of parameters to circuit switching, fast circuit switching, and hybrid switching. Even if separate voice and data networks were implemented, packet switching would be the most economical choice for each.

The study showed that further savings would result from integrating the voice and data traffic on a single network. Integration also improves the survivability of critical communication by allowing any part of the available communication bandwidth to be used for any type of information. Since the mix of information varies dynamically, only a fully integrated communication system can assure that the most essential information can always be communicated.

A further advantage of digital packet switching for voice communication, especially important to the military, is that it can be secured to any degree desired. The analog voice signals traditionally transmitted over circuit-switched lines can only be protected to a limited extent through "scrambling."

If packet switching is to meet the total DoD communication requirement, then it must also expand to support media which have not been accommodated in the past. A prime example is real-time, full-motion video. The large bandwidth required by the video signal will make it as difficult to transmit full-motion video across the wideband satellite network as it was to transmit voice across the lower bandwidth ARPANET. This will again restrict communication to one or two channels, but it is expected that packet-switched networks will continue to grow in capacity so that multiple channels of video will be be feasible in the not-too-distant future.

The switch to satellite communication introduces some problems of its own which must be investigated, specifically increased delay and higher error rates. New transmission protocols must accommodate these problems and at the same time take advantage of added capabilities, such as broadcast transmission.

7.3 GOALS AND APPROACH

The first objective of the Wideband Program as a whole is to demonstrate the feasibility of packet voice on a significant scale. Equally important is the transmission of nonreal-time data in conjunction with real-time voice to demonstrate that the functions required for data have not been lost while those for voice were emphasised. The large bandwidth of the satellite network opens new applications for data transfer in addition to providing the economic benefits of integrating the two media.

A further objective of the WBC project at ISI is to explore new modes of packet communication made possible by the large bandwidth of the satellite network.

The approach here is to develop hardware and software systems to investigate and demonstrate application of the Wideband Network in these areas. Specifically, the WBC project is working to

- Interface the Wideband Network to the commercial Switched Telephone Network (STN) to promote the widespread use of packet voice.
- Design and implement the hardware and software required for the transmission of packetized narrowband video over the satellite.
- Investigate the performance of the satellite network by itself and in combination with terrestrial networks to determine how protocols should be tuned for various network conditions and types of data.

The STN interface expands the packet voice user community by providing access via the telephone network using dial-in and dial-out facilities. This system allows packet voice to be used as an on-going experiment by a large number of people at the participating sites. It provides experience with a more heavily-loaded system than one used primarily for demonstrations. Demonstrations are important, too; the STN interface makes it possible to demonstrate the packet voice system from anywhere there is a telephone.

The objective of the WBC project's video experiments is to investigate and demonstrate transmission of video data over a packet-switched network for the first time. Real-time packet video is being developed to be used in conjunction with packet voice to enhance the effectiveness of packet teleconferencing. The goal is to build a system capable of transmitting color video with moderate motion in real time at a data rate of 1.5 megabits per second. The word "video" means television camera images at frame rates ranging from one frame or less per second (frame-grabbing) to 30 frames per second (full motion). Monochromatic video will be used initially, with color to be used later.

The Wideband Network provides an opportunity to utilize novel video bandwidth compression techniques which have been developed but not utilized because they are not well matched to fixed-rate transmission channels. Some of the best video bandwidth compression algorithms available today generate compressed data at a variable rate. A packet switched channel allows the statistical multiplexing of several variable rate streams (of voice, video or other data) to use the available bandwidth more efficiently.

The approach for video bandwidth compression algorithms is to evaluate present algorithms and adapt them to the Wideband Network, not necessarily to develop new ones. The object is to select the best current algorithm or family of similar algorithms, adapt it as necessary to optimize performance, and implement it to run in real time.

In addition to the voice and video experiments described above, the WBC project will investigate the transmission of nonreal-time data, such as large files, on the Wideband Network. The larger bandwidth of the satellite channel makes feasible bulk data transfers which probably would not have been attempted with the terrestrial ARPANET. Such data will be sent in datagram (contention) packets which fill in the bandwidth unused by the streams reserved for voice and video traffic.

To gain the full benefit of the large bandwidth of the satellite network, the file transfer protocol parameters will need to be tuned to fit the performance of the Wideband Network and the other networks involved in accessing it. For example, if the Internet Transmission Control Protocol (TCP) is used, it will require a huge window to allow several packets to be outstanding at a time. Another modification would be to return negative acknowledgments (NAKs) for packets which arrive with checksum errors in the data but not in the header. Since the Wideband Network will have a relatively high error rate and a long delay, returning NAKs could cut down the amount of time wasted waiting

for timeouts to expire. These protocol issues and others will require investigation before the Wideband Network can be used effectively.

7.4 SCIENTIFIC PROGRESS

7.4.1 Initial Network Testing

Much of the effort expended during the past year at all the Wideband Network sites has been to bring the network into operation, verify its performance, and to test and refine the transmission of multiple channels of packet speech across the network. Some of the milestones, beginning with the first integration of the network components, are listed here:

- The first packets were transmitted on the satellite channel at ISI in November 1980, during a meeting of the Wideband Program participants.
- During that same meeting, the first demonstration of cross-country packet voice transmitted using the second-generation Network Voice Protocol (NVP-II) took place. ISI's "SPEECH" packet voice system running in a PDP11/45 sent speech across the ARPANET in a loop through the Mini-Concentrator gateway running in a PDP11/45 at Lincoln and back to ISI (see Section 7.4.2).
- In January 1981, the NVP-II implementations in SPEECH and in the Lincoln Packet Voice Terminal (PVT) were tested to be compatible, and LPCM-coded speech was transmitted over the ARPANET through the Mini-Concentrator gateway at Lincoln to a PVT on the LEXNET there. The LEXNET is a coax-cable local network developed at Lincoln.
- A LEXNET and two PVTs were received at ISI in March 1981, allowing the interfacing of ISI's STN card and the PVT to begin.
- Speech packets were first looped over the satellite channel in May 1981 at Lincoln.
- In November 1981, cross-country Wideband Network transmission of speech was achieved for the first time; this was followed by an official demonstration with two simultaneous conversations, one using and STN interface at ISI.
- Slow-frame-rate video images were looped over the Wideband Network at ISI in February 1982 (see Section 7.4.4).
- A Voice Funnel multiprocessor [8], developed by Bolt Beranek and Newman Inc., was installed at ISI in April 1981. Speech was communicated across the Wideband Network to Lincoln through the Voice Funnels at both sites.
- In June 1982, the final review of the DARPA NSC program was held at Lincoln Laboratory. The review included a major demonstration of internetwork packet voice capabilities. Several calls were placed across the Wideband Network, including one to an STN-interface-equipped PVT at ISI.

7.4.2 PDP11-Based Voice Terminal

ISI's "SPEECH" packet voice system was developed during the NSC program to run on the PDP11/45 and FPS AP-120B hardware and used the original Network Voice Protocol. The WBC project converted SPEECH to use the second-generation NVP-II in late 1980. As the Wideband Network was not yet operational, the first configuration was set up for connection to the ARPANET. The ability to use a large amount of existing software enabled rapid debugging of NVP-II and provided an important testbed for the packet voice to be used on the Wideband Network.

During FY82, SPEECH was reconfigured to connect through the Mini-Concentrator gateway to

operate on the Wideband Network. The larger bandwidth of the satellite channel made it practical to add PCM voice coding to SPEECH. The PCM algorithm with speech activity detection was programmed in the FPS AP-120B signal processor and the SPEECH program to allow interoperation with other voice terminals at ISI and Lincoln using either PCM or LPC voice coding.

7.4.3 Switched Telephone Network Interface

The WBC project has designed and produced an interface to allow connection of voice terminals to the commercial Switched Telephone Network (STN). The interface allows the packet speech system to be called from any pushbutton telephone. Conversely, the system can dial out to any telephone. In this manner, a local access scheme (the telephone network) can be utilized without additional cost or effort.

Ten of the STN Interface Cards have been constructed, and two cards each have been distributed to two of the other program participants, SRI International and MIT Lincoln Laboratory. The interface is built on a 7-inch square circuit board which plugs directly into the Packet Voice Terminal (PVT) developed by Lincoln. The card can also be used stand-alone or with other voice terminal equipment, as is being done at SRI. The STN card is pictured in Figure 7-1, and a diagram of its functional components is shown in Figure 7-2. The hardware and software functions are described in more detail in [6].

The STN interface detects Touch-Tone (DTMF) pulses in the audio from the phone line and translates them into control codes delivered to a voice terminal. This allows a user of the system to specify a connection through the Wideband Network by using the pushbuttons on his phone. Likewise, the STN interface translates control codes from the voice terminal into DTMF pulses so that connection information from a remote site can be used to dial out to a local telephone. It is not necessary that the interface detect whether or not the destination phone is answered (a difficult task) because the ring-back tone can be returned through the audio channel to let the caller decide whether or not the call completed successfully.

In addition to the control functions just described, the STN card provides an analog/digital interface for the voice signal which comes into play once a connection has been established. A 2/4-wire hybrid couples the bidirectional phone line to the separate input and output lines of a PCM codec and filter circuit. The codec converts between analog signals and 64Kb/s digital streams.

Both the transmitted and received digital signals flow through the STN card's microprocessor so the data can be formatted for the interface to the PVT. The microprocessor performs other processing as well:

- A dynamic speech activity detection algorithm monitors the signal coming from the telephone so the PVT can be instructed to refrain from sending packets during silence. The algorithm is adaptive so that it can operate even with high background noise levels.
- Echoes of signals going from the PVT to the phone line are generated due to unavoidable imbalances in the hybrid and phone line. These echoes are prevented from getting back to the PVT by an echo suppression algorithm which elevates the silence threshold when non-silent data is being played out from the PVT.
- Constant-level signals (such as a dial tone) are detected in the signal coming from the phone line to determine when a connection from the STN card to a telephone has been terminated.

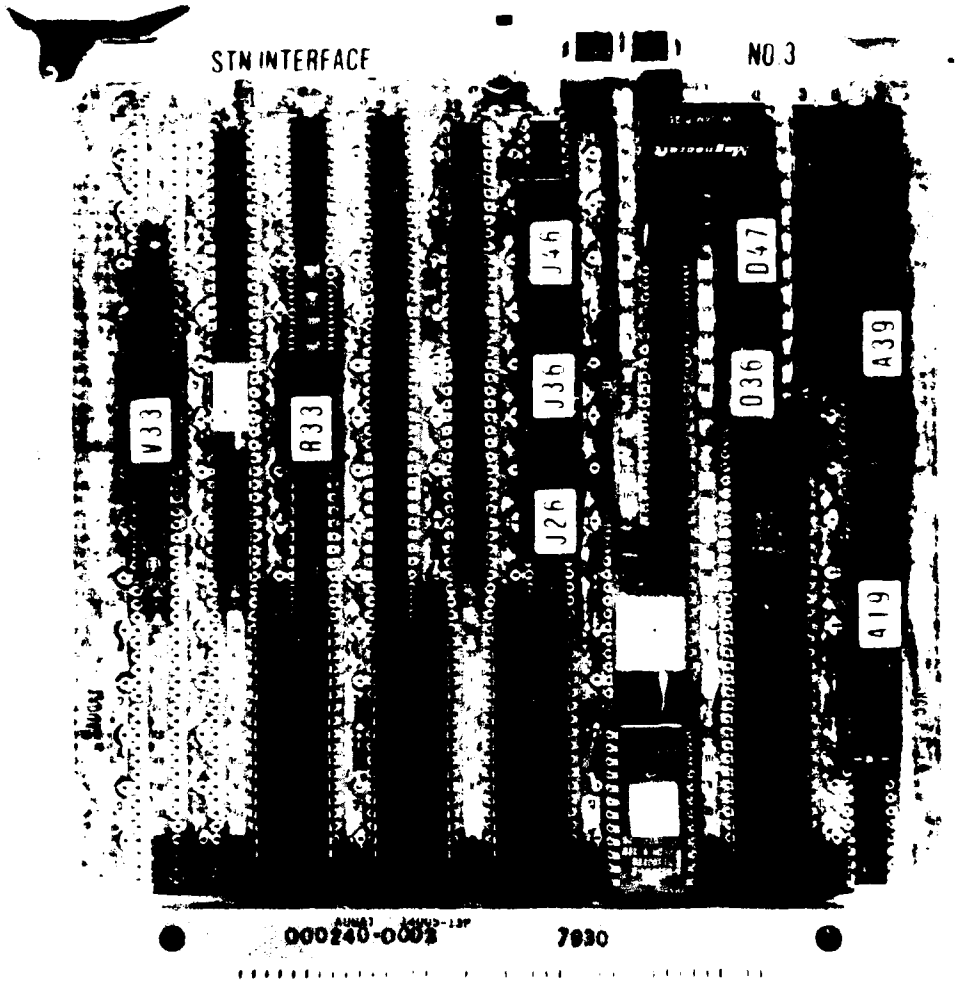


Figure 7-1: STN interface card

- Tones are generated on the phone line by playing data stored in ROM through the PCM codec in place of the signal data coming from the PVT. Tones are generated both for DTMF dialing and to provide feedback to the user on the telephone.

The STN interface can operate with low-data-rate external vocoders in addition to the PCM codec on-board. When external-vocoder mode is selected, the microprocessor transfers data between the parallel PVT interface and an asynchronous serial RS232 port on the card. The current implementation uses 8-bit binary characters on the serial line, but future plans call for the implementation of the NSC-standard low-rate serial vocoder protocol being developed by ISI and Lincoln. The serial interface was used with an LPC vocoder implemented in ISI's PDP11/FPS system to prepare for the Packet Speech Review held at Lincoln Laboratory in June 1982.

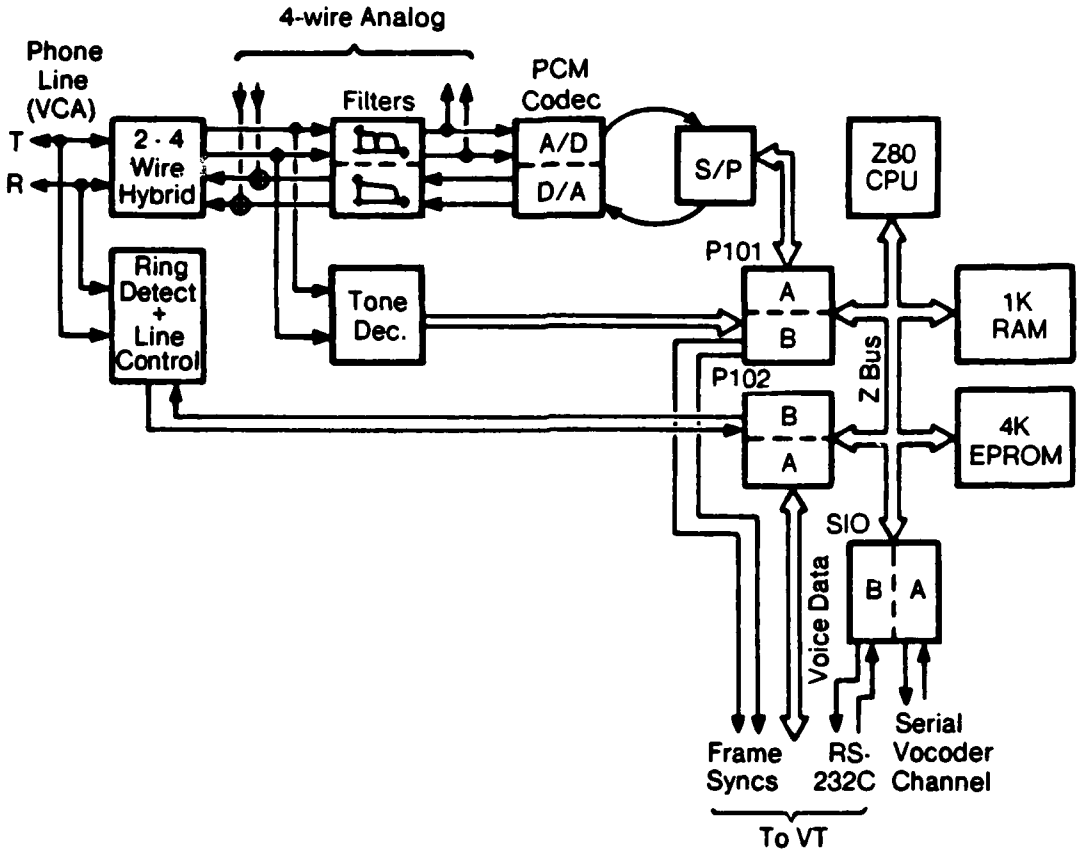


Figure 7-2: STN interface functional diagram

7.4.3.1 STN dialing sequences

ISI has issued a specification [2] for the set of dialing sequences to be used when an STN interface is called from a telephone. These sequences employ the 12 keys on a pushbutton telephone (0-9,*,#) to specify call routing through the packet voice system after the connection is established between the caller's telephone and the local STN interface.

Two methods of dialing are provided. The first allows a call to be manually routed across the Wideband Network to another packet voice terminal; then, if the terminal is equipped with an STN interface, a sequence of digits can be specified to cause the desired telephone to be autodialed. These sequences take one of the following forms, depending upon how far away the destination terminal is in the network hierarchy:

```
<EXT>*<PHONE NUMBER>#
8+1+<HOST><EXT>*<PHONE NUMBER>#
8+1+<NET><HOST><EXT>*<PHONE NUMBER>#
```

In these examples, the "+" is just a separator shown for clarity but not actually dialed, and the names in brackets indicate digit strings of various lengths (see the specification for details).

The second method allows the caller to simply dial "9" followed by the normal direct-dial long-distance telephone number of the ultimate destination, including the area code:

```
9+<10-DIGIT PHONE NUMBER>
```

When given a sequence of this form, the local voice terminal will use information in tables to automatically route the call across the packet network, choosing as its destination the packet voice terminal with STN interface that is nearest the telephone being called, then dial the necessary part of the 10-digit phone number on the remote STN interface's autodialer. This method is usually preferred because it is very simple to remember and use, but the first method is also important for experimentation with various routing possibilities.

Software to implement these dialing sequences and to support all the control functions of the STN interface has been written for the Lincoln Packet Voice Terminal. These routines have been merged into the standard program originally developed by Lincoln so that future additions to the voice terminal's capabilities can be utilized with the STN interface as well.

7.4.4 Video Bandwidth Compression

The WBC project will develop the hardware and software required for the transmission of packetized narrowband video over the Wideband Network, including a real-time video bandwidth compression system. During the past year, the following initial phases were completed.

- A literature study was conducted to evaluate the suitability of existing bandwidth compression algorithms for this application, resulting in the selection of the 2-dimensional Discrete Cosine Transform (DCT) [5] block coding family of algorithms.
- An Ikonas RDS-3000 raster graphics system was selected and installed to provide the hardware foundation for the video bandwidth compression.
- The Packet Video Protocol (PVP) was designed and documented [3].
- Software was written to support the video hardware for slow-frame-rate transmission of images across the Wideband Network using the PVP and for simulation of bandwidth compression algorithms.

Many good techniques for image bandwidth compression have been described in the literature. Block transform coding, predictive coding and hybrid methods have been considered. Block transform coding, in which the image frame is subdivided into blocks of 8x8 or 16x16 pixels which are then transformed, is the best suited to the dynamic environment of the Wideband Network. When reduced bandwidth is available, it is possible to transmit data only for the blocks that have the most change, or to reduce the high frequency information from some or all of the blocks to gracefully degrade the image quality. Block transform methods are also superior to predictive coding and hybrid methods in their ability to suffer channel errors with a minimum of image quality degradation. Errors are confined to a single block whereas they can cause noticeable steaks with predictive coding.

Other transforms, such as the Hadamard and Fourier transforms, can also be used for block transform coding, but the DCT gives provably optimum [7] compression at a slight increase in computational complexity.

A special processor to perform the DCT in real time will be designed and implemented by the WBC project. To provide a foundation for the processor, an Ikonas RDS-3000 raster graphics system has

been selected. The Ikonas provides video input and output functions and a frame buffer which can operate at full video frame rates. The system was chosen because it has high-bandwidth bus structure which allows the special compression hardware to be added in a modular fashion. One Ikonas graphics system has been procured and installed on the PDP11/45 at ISI. A second system is being procured for installation at Lincoln Laboratory to allow video transmission between the two sites.

A program called IMAGE has been written to serve as a basis for investigation of video bandwidth compression algorithms. Support routines for the Ikonas display, an AED 512 display system, and the FPS AP-120B signal processor have been developed. Using this system, video has been sampled at slow frame rates and transmitted uncompressed in a loop over the satellite channel. The DCT algorithm has also been implemented in the FPS with a simple coding scheme to allow compressed images to be sent over the channel, although still at a reduced rate.

To send the images though the network, the IMAGE program implements the Packet Video Protocol. The PVP is a set of extensions to the NVP-II voice protocol to handle the additional video medium within the same connection and conferencing structure used for voice. The PVP supports transmission of all types of real-time image data, including black and white video, color, freeze-frame, high or low resolution. It provides for the communication of functions as well as data so that algorithms can be adjusted dynamically for techniques such as embedded coding. The PVP also allows partial updates of video frames so that sophisticated algorithms can be used which update the transmitted image only when and where necessary.

The first transmissions of images across the satellite were at a slow rate of several seconds per image, but provided an important means for early testing of protocols, equipment, and channel performance. A very beneficial by-product of these tests was a visual display of channel error patterns. There has been some evidence of interference (RFI) at the ISI ground station, and the images of the error patterns caused by that interference may help explain the nature and causes of the errors.

7.4.4.1 Image dithering

To facilitate examination of multiple error pattern images, software was written to format the data for printing on a facsimile machine using facilities developed by the Internetwork Concepts research project at ISI for multi-media mail (Chapter 5). Since the facsimile machine can only print black or white, a program was developed to print dithered images. In this case, each 8-bit pixel in a video image is translated into a 3x3 cell of 1-bit (black/white) facsimile pixels; the resulting image has about ten intensity levels depending upon the number of dots which are black in the 3x3 cell. A feature is provided to add random noise to the dither threshold to avoid the contour lines which result from the small number of intensity levels.

7.4.5 WWVB Clocks

ISI has obtained four WWVB radio clocks to allow measurements of absolute transit times across networks in Wideband and Internet experiments. These clocks have now been calibrated with respect to each other and absolute standards. A small "TV Line 10" device was constructed which looks for a specific pulse within a standard broadcast television signal. By comparing the time of that pulse according to the WWVB clock with the time of the pulse published by a local institution with a

clock standard, the offsets of the WWVB clocks were determined. The clocks were also calibrated agains a travelling Naval Observatory clock. By dialing into the thumbwheel switches on the clocks the consistent offsets determined by these tests, the output of the clock achieves absolute accuracy with deviations less than ±0.5 ms.

Measurements of ARPANET packet transit times between ISI and Lincoln Laboratory were conducted using the WWVB clocks for synchronization of timing in the communicating systems. The SPEECH program (Section 7.4.2) was configured for both sites and augmented to transmit measurement timestamps with the speech packets according to the NVP-II protocol. Techniques developed for these measurements will be used to assess the performance of the Wideband Satellite Network in the future.

7.4.6 EPOS Operating System

The EPOS operating system, developed by ISI, is supporting the Mini-Concentrator gateways to the Wideband Network. EPOS has been augmented with the capability to bootstrap from the TU58 cartridge tapes which are part of the gateway systems. This allows operation without terminal access from the ARPANET, which is important for the new Wideband Network nodes at military installations which do not have ARPANET access. Device support for the RL02 disk has also been added to provide faster loading for some of the gateway computers which have the disks, such as ones being used for development at Lincoln Laboratory.

Software was added to EPOS to accurately obtain the time of day from the WWVB clocks described in the previous section. Special clock peripheral devices have been designed and built for the PDP11 computers that run EPOS to allow the time to be accurately maintained after it is set.

7.5 IMPACT

The impact of the DARPA-sponsored packet voice research has been very significant. In June 1982, the final review of the NSC program was held at Lincoln Laboratory, and a number of guests who represent potential military users of the technology were invited. The review included a major demonstration of the packet voice technology developed during the NSC program and being applied on a larger scale in the Wideband program. Several voice calls were placed using three different voice coding algorithms in a total of eight packet voice terminals of various types at three sites on the Wideband Network (ISI, Lincoln, and SRI) and including connections to the Packet Radio Network at SRI and the Switched Telephone Network at ISI. The success of this demonstration has led to requests for additional demonstrations in the future.

The ARPANET has clearly demonstrated the effectiveness of packet switching and its place in defense data communications, but a decade has elapsed in the meantime. Even though the first proposals for packet switching recommended the technology for both data and voice [1, 9], the huge fixed investment in circuit-switched voice equipment in place today means that the transition to packet-switched voice may be even slower. However, the interest of the public phone companies in packet voice is a clear indication that it will happen.

Since the Wideband effort is a cooperative program between DARPA and DCA, it provides a real opportunity to help shape military plans for future secure communications systems. Three military

bases will soon be included as sites on the Wideband Network, opening the possibility of joint experimentation. The increase in bandwidth which the satellite provides will allow experimentation not only with several voice channels but also with other media, such as video, graphics, text and facsimile. Eventually it will be possible to communicate between command centers in a teleconferencing mode for improved control of crisis situations. Even in less demanding circumstances, such as in the conduct of everyday military business, multi-media conferencing could reduce the need for travel and increase the productivity of the reduced military manpower available today.

7.6 FUTURE WORK

7.6.1 Packet Voice Experiments

Packet voice experiments on the Wideband Network will continue throughout FY83. ISI will participate in development, testing, measurement, and maintenance of the hardware and software which support packet voice. In particular, additional software to support dynamic tables for automatic call routing to STN interfaces and rotary selection among STN interfaces at any one site will be developed. These features will make it easy for the Wideband Network to be used for routine telephone calls.

Considerable effort will likely be required, as it has been in the past, to monitor the operation of the packet voice terminals, the gateways, and the network itself and to provide assistance as needed to maintain them. ISI is not just a passive consumer of the Wideband Network service, but also actively participates in bringing up the system and testing it.

7.6.2 Bulk Data Transfer

To investigate the transmission of nonreal-time data on the Wideband Network, the WBC project will develop software to offer the satellite channel as an alternative to long paths through the terrestrial ARPA network for bulk data transfers. This facility could be provided in steps:

1. A separate File Transfer Program (FTP) which transfers files over the satellite channel could be offered to users as an alternative. It might still be most reasonable to maintain the control part of the connection over the ARPANET, and only use the satellite network for the data connection.
2. After the separate FTP program has been tested, the standard FTP program could be modified to transparently make a choice of ARPANET or Wideband Network for data transfer depending on the size of the file to be transferred and the loading of the two networks.

The protocol to be used for the FTP will probably be (at least initially) the standard FTP/TCP/IP combination. Experiments will be conducted to determine how throughput is affected by the size of the flow-control window and the use of additional protocol mechanisms such as negative acknowledgments for packets containing corrupted data but intact headers.

The PDP11/45 computers being set up at ISI and Lincoln to support the packet video experiments will also serve as convenient testbeds for the FTP experiments, due to the accurate time

measurement facilities in the EPOS operating system running on these machines and the full control which is possible because these machines are dedicated.

7.6.3 Real-Time Packet Video

A major portion of the ISI WBC project effort will be the development of a real-time video bandwidth compression system for use on the Wideband Network. The goal will be to build a system capable of transmitting color video with moderate motion in real time at a data rate of 1.5 megabits per second.

Evaluation of video bandwidth compression algorithms over the past year has lead to the selection of the Discrete Cosine Transform block coding family of algorithms. The details of the final algorithm remain to be specified. Then the algorithm will be implemented in real-time hardware and software to be deployed at ISI and Lincoln in early calendar 1983. This first stage of the implementation will be for monochrome transmission only; the system will be extended to full color during the remainder of FY83.

Several variants of the cosine transform are in use. These algorithms are being tested with nonreal-time simulations in the FPS and in the PDP11 to compare their performance and to estimate the complexity of the implementing them in real time. Specific questions to be answered include the tradeoff between the block sizes of 8x8 and 16x16 pixels and the number and quantization of the transform coefficients to be transmitted.

Current plans call for interframe coding to be employed in the real-time video system. It is expected that larger gains will be achieved by determining how the interframe coding should be done to best match the packet satellite characteristics than by attempting to squeeze more performance out of a transform algorithm.

Selection of the ESI coding rate is an important part of this optimization. For a fixed channel capacity, it must determined which of the following directions improves overall quality:

1. increasing the ESI coding (e.g., to rate 1/2) to reduce the bit error rate, which requires more video bandwidth compression to fit the reduced data rate available, or
2. decreasing the ESI coding (e.g., to rate 1) to allow a higher data rate and less stringent video bandwidth compression, while suffering a corresponding increase in bit errors.

One might expect the first choice to be the better one since it employs more "intelligence." On the other hand, if the compressed data can be encoded so that it is relatively insensitive to bit errors, the second choice might give better overall quality.

Specification of the real-time system is an interactive procedure in which tradeoffs are made between performance and hardware architecture requirements. Testing of various options in nonreal time is part of the procedure. The result will be a design of the hardware structure, which will then have the details of hardware implementation filled in. Implementation of the real-time system includes hardware construction and checkout along with coding and/or microcoding for the processing elements involved. The hardware will then be deployed at ISI and Lincoln.

A high-bandwidth processor will be required to couple the packet video stream to the Wideband Network. ISI has received a BBN Voice Funnel which should be able to provide the necessary

processing power. Software will be developed for the Voice Funnel as required to attach the video hardware and process the data. After real-time video transmission is achieved, then the final adjustments and refinements can be put into place as the performance of the compression algorithm can be evaluated on image sequences including motion. In addition the control functions for video transmission must be integrated with the packet voice software.

In the second half of FY83, video transmission will be extended from monochrome to color. The algorithm will remain basically the same; the research involved is to decide how the color information should be included. Two possibilities are to use separate red, green and blue signals, or to extract luminance and chrominance information. It is also necessary to determine how to trade off color resolution against the other dimensions in the final coding.

REFERENCES

1. Baran, P., et al., *On Distributed Communications, vols. I-XI*, Rand Corporation, Research Documents, August 1964.

2. Cole, E. R., Dialing in the Wideband Network, USC/Information Sciences Institute, W-Note 26, April 1981.

3. Cole, E. R., PVP - A Packet Video Protocol, USC/Information Sciences Institute, W-Note 28, August 1981.

4. Gitman, I., and H. Frank, "Economic analysis of integrated voice and data networks: A case study," *Proceedings of the IEEE* 66, (11), November 1978, 1549-1570.

5. Jain, A. K., "Image data compression: A review," *Proceedings of the IEEE* 69, (3), March 1981, 349-389.

6. Merritt, Ian H., and Robert H. Parker, Switched Telephone Network Interface Card (STNI), 1981.

7. Pratt, W. K., *Digital Image Processing,* Wiley-Interscience, New York, 1978. [See especially chapter 23.]

8. Rettberg, R., C. Wyman, D. Hunt, M. Hoffman, P. Carvey, B. Hyde, W. Clark, and M. Kraley, *Development of a Voice Funnel System: Design Report*, Bolt Beranek and Newman Inc., Report No. 4098, August 1979.

9. Roberts, L. G., "The evolution of packet switching," *Proceedings of the IEEE* 66, (11), November 1978, 1307-1313.

8. FORMAL SEMANTICS

Research Staff:
Vittal Kini
Allen Stoughton

Consultant:
David F. Martin

Support Staff:
Lisa Holt

8.1 PROBLEM BEING SOLVED

The design and implementation of the Ada [1] programming language were commissioned by DoD with the intention of requiring most future military systems to be programmed in Ada. It is therefore necessary that Ada be precisely understood by both its users and implementers. To this end a denotational formal semantic definition (FSD) of Ada has been developed at INRIA [4]. Its intent is to provide the precise meaning of the language and its constructs via the mathematical formalism which underlies the denotational semantic descriptive technique. Due to the complexity of Ada, and despite the power and elegance of the denotational semantics method, the FSD itself is quite large: both the static (compile-time) and dynamic (run-time) phases of the definition consist of hundreds of mutually recursive functions. As a result of this inherent complexity, it is difficult for a human being to understand the definition. There are two undesirable consequences of such complexity. On the one hand one may quite reasonably assume that the sheer size and complexity of the FSD make it very likely that it contains more than a few errors. On the other hand the complexity of the FSD makes its application to practical cases very difficult even were it error-free.

8.2 GOALS AND APPROACH

One approach to understanding a denotational definition is to symbolically execute the definition on specific example programs. Attempting to do this without machine assistance will likely result in a great many errors and is, in a practical sense, impossible. Unaided human application of this FSD to understanding Ada programs is at best an arduous task. It is therefore imperative to construct appropriate tools to aid the understanding and validation of the Ada FSD. Such tools can be used in two ways. Initially, Ada test cases whose semantics are well understood can be used to test the correctness of the FSD. Subsequently, after confidence in the correctness of the Ada FSD has increased, the tools can be used to answer very specific questions about specific parts of the FSD as they relate to example Ada programs whose semantics are not readily apparent.

The Ada FSD is written in a typed lambda calculus expressed in an "Ada-like" syntax; we shall henceforth call this language AFDL, an acronym for Ada Formal Definition Language. It is necessary to build tools which will

- translate the functions and data types of the Ada FSD into an equivalent directly executable intermediate language (AFDL-IL),
- transform candidate Ada test programs (such as the Softech compiler test cases [2]) into corresponding abstract syntax trees, and
- apply the translated FSD to the abstract syntax trees to obtain via interpretation the static and dynamic semantics of the corresponding programs.

The semantics thus obtained can be compared to the expected meaning. In addition, tools to

generate useful items such as cross reference listings of the FSD's components are also indispensable aids.

In this report, we describe in more detail our approach to validating the Ada FSD. First, we briefly describe the INRIA meta-language, AFDL, and the extensions we were forced to make to it, and give an overview of the structure of the INRIA Ada FSD. Next, we describe the various tools we have built and their application to the FSD. Finally, we describe the outcome of the project.

8.2.1 Ada Formal Semantic Definition

8.2.1.1 The meta-language of the Ada FSD

AFDL, the meta-language in which the FSD is written, is an applicative language with an Ada-like syntax. The language contains function, block, conditional, and case statements, simple expressions, and packages (for modularity and information hiding) as in Ada. AFDL's basic data types are the integers and the booleans, and its data type constructors are enumerated types and (unlike Ada) function types. Conspicuously absent from AFDL are the sum (union), product (record), and sequence (array) types; these types are basic to the denotational semantics method. As a result, the data types upon which the FSD is based are only defined informally in plain language, and are not formally defined in AFDL (or any other language). The absence of formal explication of the entire base-level of the definition is one of the major impediments to the human reader who wishes to understand the formal definition (the task is akin to trying to understand a large software system in which the data type declarations were only cursorily outlined in English). This deficiency must be remedied before the FSD can be tested.

Our approach to this problem was to extend AFDL, in an upward compatible way, to include sum, product, and sequence types, along with their associated operations. This approach provides a meta-language capable of conveying the entire definition in a formal manner, and thus facilitates both human understanding and machine execution of the definition. We call this extension AFDL+. Further details of AFDL+ are provided in [5].

8.2.1.2 Structure of the Ada FSD

The Ada FSD basically consists of a collection of mutually recursive functions together with a repertoire of intrinsic basic data types. The Ada Reference Manual [1] is divided into a number of chapters, each devoted to a specific aspect or component of the language. None of the FSD appears in this manual. The Ada FSD ([4] and later versions), however, is organized by "folding" it into the Reference Manual so that each chapter contains the pertinent components of the Ada FSD. The intrinsic FSD data types and operations on them are intended to be described in Appendices to the Ada FSD in order to separate these base-level concerns from the rest of the FSD.

From an operational point of view, the Ada FSD is organized into three "phases," one of which is syntactic and the others semantic. The syntactic phase establishes a relationship between the concrete and abstract syntax of Ada by providing a specification of both the concrete and abstract syntactic domains together with a (constructive) mapping from the former to the latter. In practice, this mapping is implemented as a parse-driven construction of Ada abstract syntax trees from corresponding Ada program strings. There are two semantic phases which process abstract syntax trees. The first, called static semantics, performs what are generally considered to be "compile-time"

functions such as static type-checking and overloading resolution. This phase produces what we term an *annotated abstract syntax tree*, which is a modified form of the tree input to the phase. The annotations consist of error messages and static environment information gathered by the phase such as the overloading resolutions, visibility calculations, and normalizations. The final semantic phase, called dynamic semantics, determines the "run-time" semantics (the meaning of procedures, expressions, etc.) of error-free annotated abstract syntax trees output from the static semantics phase.

8.2.1.3 Status of the Ada FSD--August 1982

At this time, the Ada FSD is incomplete in that (excluding the semantics of Tasking in Ada which is not covered) many of the semantic functions in the chapters of the FSD document are either missing or contain errors (both type and logical), and the data types and functions in appendices of the document are largely unspecified. The functions missing from the November 1981 version of the FSD have been identified, and type errors have been detected by the use of type-checking tools developed at ISI. The results were communicated to the authors of the FSD. The burden of defining these missing types and functions and of correcting these errors necessarily falls upon INRIA.

In the FSD document, the concrete and abstract syntaxes of Ada are defined, the latter less explicitly than the former. In fact, there exist two abstract syntaxes: a post-parse abstract syntax (which is input to the static semantics phase) and a post-static-semantics abstract syntax (which is input to the dynamic semantics phase). The correspondence between the (post-parse) abstract syntax and the concrete syntax in the document is implicit; it must be given explicitly or else the reader must be given enough information to deduce the exact correspondence, as this is necessary for a detailed understanding of the FSD. INRIA has, however, supplied to ISI an LR(1) syntax for a superset of Ada, together with a preliminary correspondence between it and the Ada abstract syntax.

8.2.2 Mechanical Interpretation of the Ada FSD

In this section we shall describe the processes whereby the Ada FSD is translated into suitable intermediate forms that may then be interpreted. The interpretive execution of the FSD will allow the testing and validation of the FSD and, subsequently, the accurate determination of the semantics of given example Ada programs. Figures 8-1, 8-2, 8-3 and 8-4 illustrate these processes.

The boxes in these figures represent abstract machines that accept inputs and produce outputs in certain forms. Boxes subdivided vertically into two compartments represent abstract machines that are constructed by loading what is represented by the upper compartment into the abstract machine represented by the lower compartment. Compartments may themselves be vertically subdivided into compartments in a hierarchical manner (see Figure 8-4, for example). Numbers above the top left corner of a box uniquely identify an abstract machine and two boxes labeled with the same number therefore represent the same abstract machine.

8.2.2.1 Generation of parsers

To generate parsers for the Ada and AFDL languages we use the LR program [7] which accepts an LR(1) syntax for a language and outputs syntax tables. These tables are used to control a table-driven LR parser. Figure 8-1 shows this process for Ada and AFDL. Abstract Machine 1 is the LR program referred to above. The table-driven parser (bottom half, Abstract Machines 2 and 6) was constructed in Interlisp.

Parser Generation

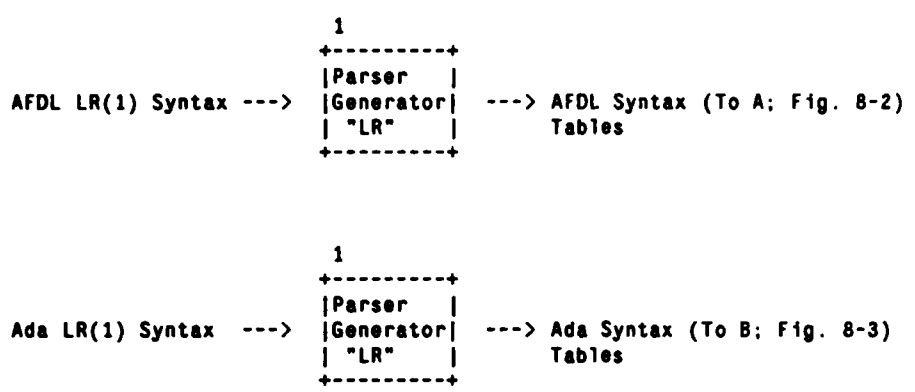
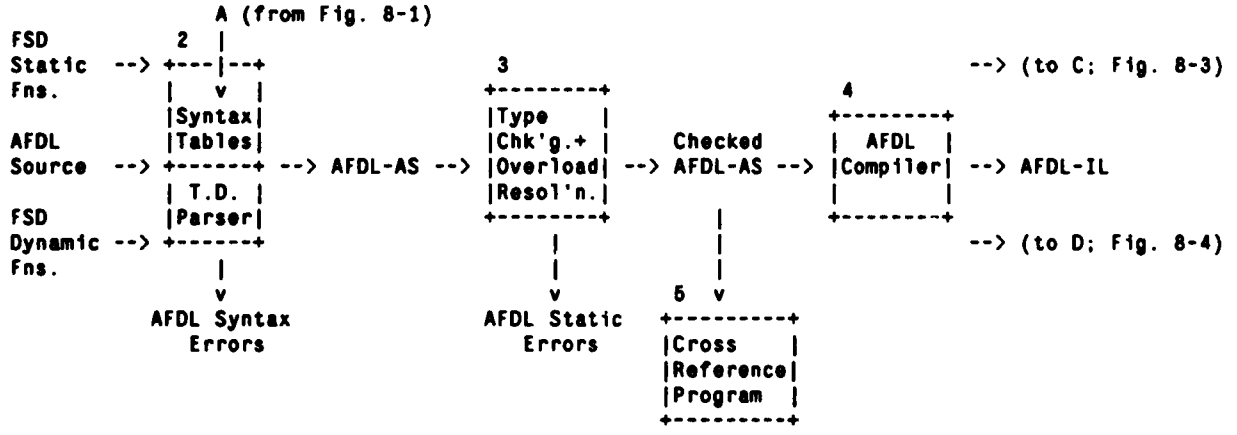


Figure 8-1

Processing FSD Semantic Functions



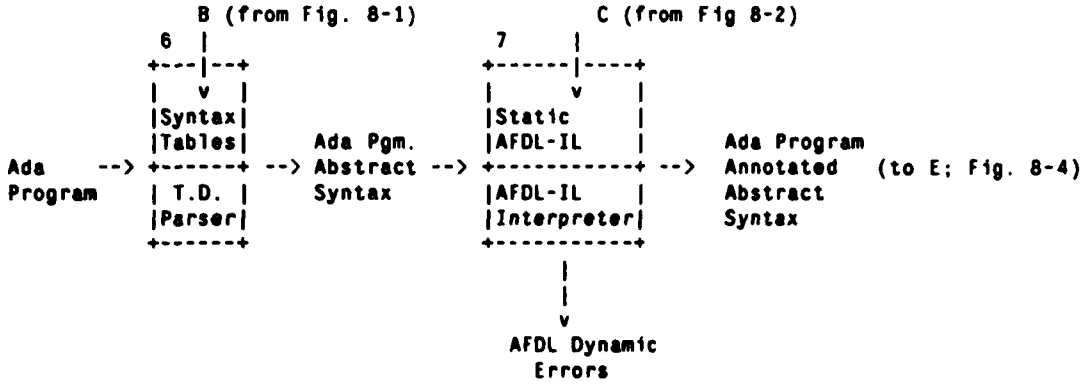
Key: T.D. Parser = Table Driven Parser

Figure 8-2

8.2.2.2 Processing of FSD semantic functions

The static and dynamic semantic functions of the FSD, written in AFDL, are translated into a directly executable intermediate language, AFDL-IL, by the process depicted in Figure 8-2. Abstract Machine 2 parses the AFDL text of the functions and produces the AFDL abstract syntax (AFDL-AS)

Static Semantics (PASS 1)



Key: T.D. Parser = Table Driven Parser

Figure 8-3

Dynamic Semantics (PASS 2)

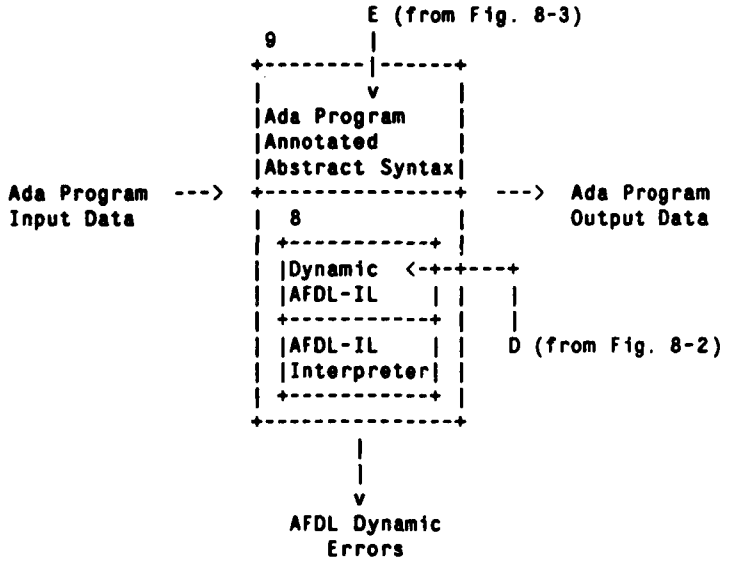


Figure 8-4

form of those functions (the AFDL text is first extracted from the surrounding prose in the FSD source document, using a SNOBOL program). Errors discovered at this stage include AFDL syntax errors arising from misspelled or missing keywords or delimiters, or from improperly constructed AFDL syntactic forms. The syntax tables generated by the LR program in the process of Figure 8-1 are loaded into the table-driven LR parser to yield Abstract Machine 2.

The output of Abstract Machine 2 is then fed to Abstract Machine 3, which checks for incorrect AFDL data type usage and resolves any overloading of function names. Errors trapped at this stage consist of improper type usage, unresolvable function name overloading due to ambiguity, missing declarations, misspelled declaration names leading to the appearance of missing declarations, etc. The output of Abstract Machine 3 is the AFDL-AS form of the static and dynamic semantic functions in which such errors have been eliminated. The actual output of the type-checker/overloading-resolver is a symbol table that relates all symbols in the global scope of the definition to their type-checked values. Thus, for example, the value of a symbol representing a typename will be the definition of that type, and the value of a symbol representing a function will be the type of that function along with the abstract syntactic form of the function body.

For efficiency, the abstract syntax tree forms of the semantic functions are not directly executed during the interpretation process. A compiled form is used instead. Abstract Machine 4 compiles the AFDL-AS functions, producing instruction sequences for the AFDL-IL interpreters depicted in Abstract Machines 7 and 8.

8.2.2.3 Cross reference facility

Abstract Machine 5 in Figure 8-2 represents the cross reference facility. This program accepts the AFDL-AS output from Abstract Machine 3 and produces a listing of attributes for each symbol in the global scope of the FSD. Thus, for instance, if a symbol represents a semantic function, the attributes include a list of functions it calls and a list of functions that call it. These cross-reference lists are a useful tool in checking the FSD for irregularities.

8.2.2.4 Static semantics (PASS 1 of the Ada FSD)

The Ada FSD is constructed to provide the semantics of an Ada program in two passes that evaluate, respectively, the static and dynamic semantics of the program. The process of Figure 8-3 depicts the first, or static semantics, phase. The Ada programs whose semantics are to be evaluated are first rendered into an abstract syntactic form by an LR(1) Ada parser (Abstract Machine 6). This abstract machine is constructed by loading the syntax tables for Ada, produced by the LR program, into the table-driven parser mentioned in Section 8.2.2.1.

The Ada program abstract syntax is fed into Abstract Machine 7 which does the actual static semantic evaluation. Abstract Machine 7 is constructed by loading the compiled AFDL-IL form (symbol table) of the static semantic functions of the FSD (output from Abstract Machine 4, Figure 8-2) into an interpreter for AFDL-IL. The interpreter is a stack machine with a storage retention strategy programmed in Interlisp. Abstract Machine 7 is, therefore, the static semantics evaluator and embodies the corresponding part of the FSD.

The output of Abstract Machine 7 is an annotated abstract syntactic representation of the Ada program that was input to the process of Figure 8-3. The annotations correspond to the error messages and static environment information that were obtained from the program and checked for

correctness by the static semantic functions of the FSD. By using this annotated abstract syntax representation, the dynamic semantic functions in PASS 2 need not re-analyze the Ada program to obtain static environment information.

During the running of Abstract Machine 7, AFDL dynamic (run-time) errors may occur (e.g., selection of a non-existent element of a sequence type (array)). Such errors are defects in the FSD and must be corrected. In addition, various errors may also be detected by the *FSD itself* and recorded as annotations in the abstract syntax tree produced. For instance, unresolvable overloading of Ada function or operator names, ambiguities due to importation of declarations by means of **use** clauses, and improper data-type usage, are errors that may be statically determined to exist in a program. The flagging of such errors, however, may signify different things depending on the intended use of Abstract Machine 7. In initial phases when the FSD is being validated, it is supposed that the semantics of the test-case input Ada programs are well understood and are known to conform to program specifications. In such cases, the errors flagged by Abstract Machine 7 will in all likelihood be caused by errors in the FSD static semantic functions themselves. In latter phases, subsequent to the validation of the FSD, such errors flagged by Abstract Machine 7 will point to errors in the Ada programs themselves. In such cases, the FSD static semantic functions will have fulfilled their intended purpose.

8.2.2.5 Dynamic semantics (PASS 2 of the Ada FSD)

The second pass of the Ada FSD, the dynamic semantics phase, is depicted in the process of Figure 8-4. It is seen from the figure that Abstract Machine 9 is constructed from Abstract Machine 8 by loading into the latter the annotated Ada program abstract syntax representation that was output by Abstract Machine 7. Abstract Machine 8 is itself constructed in a manner similar to that of Abstract Machine 7. This is done by loading the compiled AFDL-IL (symbol table) form of the FSD dynamic semantic functions (output of Abstract Machine 4 in Figure 8-2) into the AFDL-IL interpreter. The result is an abstract machine with the capability to evaluate Ada program annotated abstract syntax trees.

The dynamic semantics interpreter (Abstract Machine 9) is obtained by loading Abstract Machine 8 with the annotated abstract syntax representation of the Ada programs that are the output of the static semantics evaluator (Abstract Machine 7). Machine 9 is an executable object that embodies the semantics of these Ada programs. It accepts representations of the input data that the Ada programs would have accepted and through a process of interpretation produces the output data of those programs (assuming they terminate). In addition to viewing the output data produced by Ada test programs, it is also possible to view the internal computation states of those programs. Both the external and internal behavior can be used in judging the correctness of the FSD and Ada programs.

In a manner similar to that described in Section 8.2.2.4, AFDL dynamic (run-time) errors can occur during the execution of Abstract Machine 9. These are FSD errors. In addition, the FSD dynamic semantic functions themselves may detect run-time errors in the Ada test programs. These errors are indicated as part of the output data of those programs, and may represent errors either in the dynamic functions of the FSD or in the Ada programs themselves. Furthermore, the program output data produced by Abstract Machine 9 may be examined to determine whether it is as expected. Again, in the case of test-case Ada programs with well-understood semantics, incorrect output data is symptomatic of errors in the FSD which will require identification and correction.

8.2.2.6 Summary

It may be observed from the foregoing description that the processes depicted in Figures 8-1 and 8-2 are performed once for each version of the Ada FSD (except for the process in Figure 8-1 which generates the syntax tables for the LR(1) Ada grammar; this is done precisely once, unless the grammar for Ada changes, a much less likely event). Here, a new version of the Ada FSD is created each time errors detected in the FSD are corrected. Therefore, these processes are "constructor processes" in that they are used to construct the Ada FSD validation tools. When errors in the FSD are detected during the running of these constructor processes, they must be terminated prematurely, a new version of the FSD generated, and these processes restarted.

On the other hand, the processes depicted in Figures 8-3 and 8-4 are run for each test-case Ada program whose semantics is to be evaluated, or whose known semantics is to be used to validate the FSD. As such, these processes may be viewed as "validation processes."

In summary, the processes of Figures 8-1 and 8-2 are used to parse, type-check and prepare intermediate forms of the FSD static and dynamic semantic functions. The checked intermediate forms are then loaded into the AFDL-IL interpreter to produce the static and dynamic semantics interpreters. These interpreters are used in the processes of Figures 8-3 and 8-4 to evaluate the semantics of test-case Ada programs and validate the FSD.

8.3 SCIENTIFIC PROGRESS

The software tools we have described in this report have been constructed and are fully operational at USC/Information Sciences Institute. An exception is the user-interface to the AFDL-IL interpreter which must perforce remain in a rudimentary state until experience with exercising the complete INRIA Ada FSD is gained. For the moment we have the capability to set conditional breakpoints upon entries to and/or exits from semantic functions and also at the level of individual instructions in the compiled forms of the semantic functions. The full power of Interlisp is available at a breakpoint, and thus it is possible to observe the static and dynamic chains, variable bindings and continuations in the interpreter. This ability to intervene interactively in the interpretation is a feature that distinguishes our work from that of Mosses [6].

An extensive list of type errors generated by the type checker (Abstract Machine 3) and cross reference listings have been supplied to INRIA for the entire FSD. Unfortunately, because the FSD is currently incomplete, it has not yet been possible to test the static and dynamic interpreters on the FSD or to begin validating the FSD. It is hoped that efforts at INRIA will eventually remedy this situation. In the meantime we have employed the denotational definition of Gordon's example programming language TINY [3]. Our version is essentially a transliteration into our metalanguage AFDL + of the *continuation*-style semantic definition of TINY that Gordon provides. In this definition, all of the static and semantic domains are defined in terms of the basic types *INTEGER* and *BOOLEAN* and the various domain constructors. We have used this definition to exercise our tools.

We have succeeded in processing the TINY definition through the parser, type-checker and compiler. The compiled definition has been run on the interpreter with several examples meant to test the various paths in the definition. The largest TINY program we have executed under the AFDL interpretation of the definition has been one that accepts integers from the input stream and produces the corresponding factorials in the output stream. Since the TINY language does not have a

multiplication operation, this was implemented via iterative addition! While such a program is quite small, the work done by the interpreter in terms of semantic function applications and the corresponding manipulation of static and dynamic chains is not trivial. Thus it has proven to be a reasonably good test of the software.

We have extended INRIA's semantic metalanguage AFDL to a more generally applicable semantic metalanguage, AFDL+. Since AFDL+ contains the necessary idioms for writing denotational semantic definitions, we expect that our software could also be useful to the scientific community at large.

8.4 MILITARY RELEVANCE AND IMPACT

Since the programming language Ada [1] was commissioned by DoD with the intention of requiring most future military systems to be programmed in Ada, it is necessary that Ada be precisely understood by both its users and implementers, in order to ensure the quality of systems written in Ada. In particular, since DoD must control Ada compiler implementations, a precise, well-structured, and validated formal definition of Ada can provide one of the principal standards to which these implementations must adhere. Beyond such considerations pertaining to particular programming languages such as Ada, good generic research toward the design and implementation of tools and methodologies for supporting the development of precise, readable, and accurate formal definitions has considerable relevance to the broader goals of understanding large programs and verifying their correctness.

The USC/ISI Formal Semantics project, which concludes at the end of FY82, has addressed the question of building tools to mechanize and validate the formal semantic definition of the Ada language. The project has accomplished the construction of such a set of tools which may be used to assist the continuing development of the Ada Formal Semantic Definition at INRIA. Upon completion of development work on the Ada definition, these tools may be used to assist readers and users of the definition. A key feature of these tools is the ability to use them in an interactive environment. Furthermore, the tools are not restricted to processing the Ada definition alone; they may be used to manipulate formal definitions of other programming languages written in the same metalanguage. They are thus capable of assisting in pedagogical settings and in generic research towards making formal semantic definitions of programming languages easier to construct and more practical to use.

REFERENCES

1. *Reference Manual for the Ada Programming Language (Proposed Standard Document)*, United States Department of Defense, 1980.

2. Goodenough, J. B., and J. R. Kelly, *Ada Compiler Validation Capability: Long Range Plan*, Defense Advanced Research Projects Agency, Waltham, Mass., Technical Report 1067-1.1, February 1980.

3. Gordon, M. J. C., *The Denotational Description of Programming Languages: An Introduction*, Springer-Verlag, New York, 1979.

4. Donzeau-Gouge, V., et al., *Formal Definition of the Ada Programming Language (Preliminary Version for Public Review)*, INRIA, Le Chesnay, France, November 1980.

5. Kini, V., D. Martin, and A. Stoughton, "Testing the INRIA Ada Formal Semantic Definition: The USC-ISI Formal Semantics Project," in *AdaTec Conference Proceedings*, pp. 120-128, Association for Computing Machinery, October 1982.

6. Mosses, P. D., *SIS: A Compiler-Generator System Using Denotational Semantics (Reference Manual)*, Department of Computer Science, Aarhus University, Denmark, Draft Report 78-4-3, June 1978.

7. Wetherell, C., and A. Shannon, "LR--Automatic Parser Generator and LR(1) Parser," *IEEE Transactions on Software Engineering* SE-7, (3), May 1981, 274-278.

9. QPRIM

Research Staff:
Louis Gallenson
Joel Goldberg

Support Staff:
Lisa Holt

9.1 PROBLEM BEING SOLVED

The QPRIM (QM-1 Programming Research Instrument) effort has produced an online interactive emulation facility housed in an existing mature operating system. The resulting emulation system is designed for use in both emulation program development and emulator-based target program development. QPRIM is also designed as a standard vendor product, capable of being acquired and maintained via normal procurement channels.

Systems for developing computer programs ideally provide a programmer, or a programming team, with a complete program-development environment that includes a complete set of tools. Unfortunately, the testing and debugging of programs created for embedded systems is typically performed either on the actual machine involved or on an emulator housed in a separate emulation facility. In either case, the program execution environment is different from--and typically far more primitive than--the program development environment. QPRIM aims to bring this program execution and testing into the programmers' working environment without paying the often prohibitive cost involved in utilizing a simulation program on the development host.

9.2 GOALS AND APPROACH

The PRIM project [2, 6] built such a prototype facility within the TENEX operating system; that facility was operational at ISI from 1974 until 1979. Unfortunately, PRIM suffered from two major defects. First, the emulation host chosen was the Standard Computer Corporation MLP-900 [14]. This particular machine was developed as the prototype for a product that Standard subsequently abandoned, leaving our MLP-900 a white elephant. Second, since the MLP-900 was designed with speed of emulation as the foremost consideration, the instruction set that emerged was rather idiosyncratic. Consequently, emulators tended to be messy to write and even worse to maintain. GPM [12], the compiler and language used to generate MLP-900 microcode, helped somewhat to hide the mess. Even so, PRIM emulators were both expensive to create and, due to the uniqueness of the MLP-900, of limited utility.

QPRIM is a PRIM-like emulation facility running under the DEC TOPS-20 operating system and utilizing a production emulation engine, the Nanodata QM-1. In addition to assembly language, QPRIM emulators may be written in SMITE, a computer description language developed at TRW, or in a variant of ISPS. Under QPRIM, the QM-1 is a new, sharable TOPS-20 resource that is available to TOPS-20 user processes (and, thereby, to TOPS-20 users) to run emulation jobs. Each user process utilizing the QM-1 does so independently of all others; the procedures for constructing and running such an emulation job are quite simple.

9.2.1 System Description

QPRIM was built and runs under the TOPS-20 operating system on a DECSystem-20. QPRIM consists of three major components: the emulation facility per se; a large interface program that gives a user (at a terminal) interactive access to the emulation facility and provides a framework for the running emulator; and the set of target machine emulators that have been built and made available.

The emulation processor in QPRIM is a Nanodata QM-1 [10]. It features two levels of firmware for implementing the user's target (or macro) machine: a low-level 360-bit horizontal "nano" machine and a higher level 18-bit vertical "micro" machine. Much of the potential power of the QM-1 is derived from the ability to tailor the micro machine to the application through the creation of appropriate nanoprograms. Nanodata has defined and implemented a base microinstruction set of some 80 instructions (out of an opcode space of 128 microinstructions) called MULTI [9]. Most applications of the QM-1 use a microinstruction set that is an extension of MULTI, and certain of these extensions have gained considerable distribution.

The QM-1 has been produced in modest numbers and is still being manufactured; it has been used for production emulators, research in emulation, and some classroom work. There is, therefore, a community of people who are already producing emulator code for QM-1s. In addition, there has been work on the production of QM-1 microcode from high-level machine descriptions. Currently, there are two languages, and accompanying compilers, that produce QM-1 microcode. The SMITE computer description language [13] is a TRW product that is supported and publicly available; much of the extant QPRIM emulation code has been written in SMITE and compiled on the RADC Multics system. The QISPS system [3] is an Aerospace Corporation effort based on a variant of ISPS [1] as the source language; it is currently used only by its creators.

9.2.2 The Emulation Facility

As an emulation engine, the QM-1 is slaved to a DEC TOPS-20 system to become its emulation resource. There are three new components that together create this TOPS-20 emulation facility:

- a hardware interface from the QM-1 to the DECSystem-20;
- a TOPS-20 monitor addition, the QM-1 driver, to manage the QM-1 and to provide process access to the QM-1; and
- a firmware emulation-control system resident on the QM-1, the microvisor.

The interface provides the physical connection between the two machines, while the QM-1 driver provides program access to the QM-1 from TOPS-20, and the microvisor supervises emulation job execution to maintain its integrity and that of the TOPS-20 system.

The interface unit that connects the QM-1 to the DECSystem-20 must meet the requirements both of QPRIM, to support the emulation facility, and of general system maintainability. For the latter, it must provide a manual disconnect function that logically separates the QM-1 from the DECSystem-20 and precludes the possibility of one interfering with the operation of the other. Second, for ease of procurement, the interface must be a standard vendor part, available for purchase and subject to normal maintainance. Third, the emulation facility requires that the QM-1 and its microcode have (shared) access to the memory of the DECSystem-20; all QM-1 memory references to the DECSystem-20 memory must use virtual addresses whose translation to real physical addresses is controlled, ultimately, by the TOPS-20 monitor. In addition to shared memory, there is need for a control communication path between the driver and the microvisor.

After discussions of various options and designs, both internally and with DEC and Nanodata, we selected an interface design [4] in which the memory access path is through an independent port into the DECSystem-20 memory and the control path is routed through the 20's IO Bus. With this design all hardware changes and interfaces lie outside the DECSystem-20 and, therefore, in the QM-1 environment. This arrangement had the additional advantage that Nanodata seemed far more interested than DEC in participating in this development and following the interface, and the entire concept, to fruition.

The microvisor comprises all the system firmware that runs on the QM-1 to support QPRIM. It consists of nanocode that defines the actual microinstruction set, plus supervisory microcode that defines the operating environment for all QPRIM emulation jobs. The final microvisor specification [7] provides a fixed microinstruction set, based on Nanodata's MULTI, sufficient to allow the execution of emulators written in the SMITE computer description language. The nanocode to support this instruction set is similar to the nanocode that supports it in standard QM-1 systems with one exception: the main store instructions have been completely reimplemented to treat the main store address as a virtual address within the virtual address space assigned to the emulation job. The main store nanocode also implements, as a dynamic option, reference break detection utilizing the two most significant bits of each main store word as break flags; when this option is active, the executing microcode sees a 34-bit (rather than 36-bit) memory. The remainder of the microvisor is microcode that executes on the QM-1 in supervisory mode; the two halves of that code are concerned with emulation job control (including job swapping and page table management) and with intrajob task scheduling in the multitask job environment that is supported by QM-1. An emulation job can be interrupted after any microinstruction, swapped out, and eventually swapped back in to the QM-1 with full transparency; no job status or other information is retained in the QM-1.

The TOPS-20 operating system support for QPRIM consists of the addition of a new device type (known as "QM1:") and a driver in the TOPS-20 monitor to support the device and control access thereto. The driver implements those monitor calls (amongst the existing device-dependent monitor calls) that are needed by a program to create and control a QM-1 emulation job; it also prescribes the format of the main store and context of that emulation job. Main store is a normal TOPS-20 process virtual address space (a fork) created by the program. The context is a contiguous memory area (within the program itself) containing an image of the emulation job's control store; the context is copied into, and back out of, the real QM-1 control store by the microvisor as part of its job-swapping function. When the job is not running, the context, as well as the main store, may be examined, and modified arbitrarily, by the controlling program; while the job is running (in the QM-1, of course), only the main store is accessible to the program.

9.2.3 The QM1 Program

The interface program, known as QM1, is that part of QPRIM visible to the user. It is a large interactive command processor that knows how to create and control a QM-1 emulation job on behalf of its (human) user at a terminal. (While the emulation facility is a normal system resource available to any user process wishing to access it, QM1 is the only such program that currently exists; each incarnation of this program creates a process that runs one emulation job, with no knowledge of, or interaction with, any other emulation job.) The interface program is the single part of QPRIM taken directly from the earlier PRIM system. Only a small part of this program is concerned with the details of a QM-1, as opposed to MLP-900, emulation job; by far the greater part of the program--and of the effort invested in writing the program--went into the command processing, the debugger, the emulation job's I/O server, and the management of data and tables concerning all of these.

The change of emulation engines by itself involved only the rewriting of two program modules: one responsible for direct communication with the QM-1 driver to control the emulation job, and the other responsible for loading emulator object code from appropriate object files into (the allocated image of) control memory. These two modules reflect the change from MLP-900 to QM-1 and from GPM (the MLP-900 high-order MOL) to SMITE.

In addition, we took advantage of the microvisor's ability to handle a multi-task emulation job. We created a manager task, which is a small, fixed program that QM1 automatically loads along with the emulator being used (by either emulator writer or target programmer). The result is an emulation job consisting of two tasks: the manager as parent and the emulator proper as sole offspring. The manager collects all detected breakpoints in a breakpoint buffer known to the debugger and signals the emulator to stop at the top of its cycle when a break occurs. In addition, reference breaks in the emulator's main store are detected by the QPRIM nanocode and reported directly to the manager for collection; the emulator is not explicitly involved. The emulator itself defines and detects the occurrence of target event breaks; each break is reported immediately to the manager. The reporting mechanism has been designed to fit cleanly into SMITE's port construct: each defined event in the emulator consists of a declared flag (for enabling the event) and a port (for reporting an event occurrence) plus a simple conditional statement located at the appropriate point in the emulator code.

9.2.4 The Emulators

QPRIM is designed to support a set of emulation tools, letting target programmers directly use the various tools--with no knowledge of QM-1s and very little of QPRIM. In order to integrate an emulator into the QM1 program environment, the emulator writer must follow a set of rules in writing the emulator and must provide necessary syntactic and semantic information to QM1 in the form of a set of tables. The tables allow QM1 to know the names and attributes of all the target machine constructs of interest to the target programmer, while hiding the details of their residence within the address space of a QM-1. Both the rules of behavior and the nature and format of the tables are described in the *QPRIM System: Tool Builders' Manual* [5], which is the primary QPRIM user document. The behavior rules are very similar to those of PRIM (though somewhat simpler because the manager now handles the task of breakpoint logging and reference break detection). When a target-machine programmer runs the program associated with his target machine, he gets the QM1 program with his emulator and tables already loaded and ready to go.

For an emulator writer who is developing and testing an emulator, QM1 has a QM-1 pseudo-emulator that consists of tables describing the QM-1 resources of interest. The emulator writer using this pseudo-emulator has available the full QM1 command processor and its facilities. He runs QM1 directly and commands the program to load his emulator (and tables, should he not want the default pseudo-emulator).

The emulator writer developing a new emulation tool to be installed in QPRIM typically begins with the QM-1 tables. As his emulator begins to take shape and work, he will want to expand the QM-1 tables to include many of the constructs of his target machine. By the time he is done, his tables are approximately the sum of the original QM-1 table and the table required to accompany his emulator for use by others. The final step in the addition of this new emulator is to move (or copy) the emulator code and tables to their final place and create the dummy program that the target programmer will call upon to run QM1.

While we do not intend (or want) to be the principal emulator writers for QPRIM, we feel that it is necessary for us to write and use some emulator code as a part of the process of QPRIM testing and shakedown. We have therefore chosen to implement two emulators (both of which were also PRIM emulators). The first is a bare Intel 8080 processor (CPU only--no IO or environment of any kind). The emulator, written in SMITE, was taken from an example in the SMITE language manual and modified to follow the QPRIM rules; the tables were copied from the PRIM version of the 8080. The total effort required was not much more than a day. The second is an AN/UYK-20 emulation, complete with optional instructions, IO channels, and a modest complement of IO devices. This latter effort comprised a substantial portion of this past year's work.

9.3 PROGRESS

This past year saw the completion of the remaining deliverables and of the QPRIM project itself. The items completed during this year were the final installation of the second QPRIM system at RADC, the AN/UYK-20 emulation tool, and the primary user manual. In addition, numerous small to modest improvements were made to the QM1 program, usually in response to requests from the early users, primarily at RADC, of the system.

Nanodata delivered and installed a second copy of the QPRIM hardware interface unit to RADC last year, but due to numerous problems, the system was not functional until April 1982. Although many of the problems were Nanodata's, the item that took the most time to fix turned out to be a problem with the DEC memory. One of the TOPS-20 memory units had a timing bug that TOPS-20, by itself, never uncovered; many months passed before DEC was convinced that the problem really resided in their hardware and fixed it. Beginning in April, then, the system began to be exercised and has proved to be as reliable as the ISI system. Except for slight driver changes that reflect the different TOPS-20 monitor versions in use at the two sites, ISI and RADC are running the same QPRIM software. The set of installed emulation tools also differs at the two sites.

An AN/UYK-20 emulation tool for QPRIM was implemented during this year. The emulator was created through manual translation of the old GPM emulator for PRIM into MULTI, the QM-1 assembly language. The latest version of the user's handbook [11], which was not available when the GPM original was written, was consulted throughout, and a number of errors (or, possibly, changes in the AN/UYK-20 specification!) in details of AN/UYK-20 instruction implementation were detected and corrected. Surprisingly, none of the changes affected the ability of the emulator to pass the AN/UYK-20 CPU diagnostics. The emulation includes the UYK-20 CPU (with the MathPac option), IOC, clock, and interrupt system, as well as a modest complement of peripheral devices; the "Level 2" software system for the UYK-20 runs on the emulator. An on-line document has been written to aid the new user in learning the attributes of the tool.

The *QPRIM System: Tool Builders' Manual* [5] was finished and published in October 1981. This is the complete QPRIM manual for the person desiring to create a QPRIM emulation tool, covering all aspects of the QPRIM tool construction process. Preliminary versions of the manual were distributed to the pioneer QPRIM users for their use; their comments and unanswered questions led to numerous editing changes and provided the impetus for much of the added examples and explanatory text.

9.4 IMPACT

The two QPRIM systems, at ISI and RADC, have developed a modest user community that appears to be growing, albeit slowly. The RADC group has introduced QPRIM to new users at a number of Air Force Air Logistics Command (ALC) sites. RADC and the various ALCs have produced, or are currently working on, emulators for the Z8000, M68000, 6502, Nebula, 1750, and AP101-C.

A first step has been taken at RADC to provide a source of assembled code for any and all QPRIM target machines. RADC runs the NASA/McDonnel Douglas Meta Assembler [8]; it is a program that, given a description of an assembly language and of the associated instruction and word formats, becomes an assembler for the described language. Many of the emulators that are being implemented for QPRIM have already had their target assembly languages described for this assembler. Mark Pratt, at RADC, has written a small program, QLINK, to take meta assembler output and turn it into the format used by QPRIM to load the target memory of an emulation job.

We believe that QPRIM should become an important tool for a number of systems and packages being developed, in particular the National Software Works (NSW) and the Software Architecture Evaluation Facility (SAEF). Both these systems have a place for an emulation tool embedded in the offered programming environment. The future utility of QPRIM should be further enhanced by the relative ease of production of new QPRIM emulators and emulation tools, and by the availability of the meta assembler to produce loadable target code for the various emulators.

REFERENCES

1. Barbacci, M. R., G. Barnes, R. Cattell, and D. Siewiorek, *The ISPS Computer Description Language*, Carnegie-Mellon University, Computer Science Department, 1979.

2. Britt, B., A. Cooperband, L. Gallenson, and J. Goldberg, *PRIM System: Overview*, USC/Information Sciences Institute, RR-77-58, March 1977.

3. Crocker, S., QISPS is an Interlisp-based program that is currently undocumented. (Private communication.)

4. Gallenson, L., and J. Goldberg, TOPS-20 (TENEX) QM-1 interface specifications, 1978.

5. *QPRIM System: Tool Builders' Manual*, USC/Information Sciences Institute, WP-20, 1981.

6. Goldberg, J., A. Cooperband, and L. Gallenson, "The PRIM system: An alternative architecture for emulator development and use," in *Proceedings, Tenth Annual Workshop on Microprogramming*, ACM SigMicro and IEEE TC-Micro, Niagara Falls, N. Y., October 1977.

7. Goldberg, J., QPRIM microvisor specification, 1980. MMPE Note # 14 (revision 6).

8. *Meta Assembler User's Manual*, McDonnell-Douglas Astronautics Company, MDC G5876, August 1975.

9. *MULTI Micromachine Description*, Nanodata Corporation, Buffalo, N.Y., March 1976.

10. *QM-1 Hardware Level User's Manual*, Nanodata Corporation, Buffalo, N.Y., September 1979. Third edition.

11. *User's Handbook for the AN/UYK-20(V) Computer*, Naval Sea Systems Command, NAVSEA 0967-LP-598-2030 (Change 8), 1981. See Volume III, Part 5, "Hardware Description."

12. Oestreicher, D., *A Microprogramming Language for the MLP-900*, USC/Information Sciences Institute, Technical Report RR-73-8, June 1973.

13. *Advanced SMITE Training Manual*, TRW Defense and Space Systems Group, 32584-6015-RU-00, September 1979.

14. *MLP-900 Multi-Lingual Processor Principles of Operation*, Standard Computer Corporation, May 1970.

10. INTERLISP

Research Staff:
Dan Lynch
Dave Dyer
Don Voreck
Ray Bates

10.1 PROBLEM BEING SOLVED

This project is establishing an ARPA support center for Interlisp in order to provide a large address-space version for the VAX, and is designing a portable version for migration to new, more cost-effective machines as they become commercially available.

10.2 BACKGROUND

A serious problem has arisen within the research community as the research projects using LISP have developed larger and more complex programs. A basic limitation of the PDP-10, on which current systems are implemented, has been reached. The address space available (256K words) is too small; current research projects routinely require several times as much space. A recent ARPA-sponsored workshop on symbolic computation estimated that minimum requirements are 8 million addressable words and that these requirements are expected to grow towards 2↑30 words; conventional wisdom estimates this growth at .5*bits/year.

Experience has consistently shown that programming costs rise exponentially as resource limitations are approached. Thus, many LISP-based research projects are expending considerable resources in system programming efforts to mitigate the effects of this address-space limitation.

This difficulty provided the incentive for the community to consider various alternatives. The first question was whether LISP was the best language for such large, complex research projects. The result of this discussion was the conclusion that there is no existing widely acceptable alternative (although this is an important research area). The second question was which dialect of LISP is best. There are two main dialects, Interlisp and Maclisp, which differ mainly in their control structures and the environment developed around the language. The user communities expressed strong preference for their own dialects. The Maclisp community is pursuing a number of alternatives, including two new dialects: CONS, implemented on new hardware (a specially designed LISP machine), and NIL, being implemented on an S1.

ISI's Interlisp project addresses the needs of the Interlisp community. For this community, the LISP machine option was rejected because of concern for the availability and maintainability of the hardware and because of dialect incompatibilities. Instead, the VAX computer was chosen as the initial machine for Interlisp implementation on the basis of its expected widespread use throughout the academic community and the opportunity for technology transfer such widespread use affords.

It was further argued that no one machine would (or should) dominate Interlisp (or in fact IPTO) usage in the future as the PDP-10 has until now. Instead, new, more cost-effective machines will be

appearing with increasing frequency. Therefore the new, large-address-space Interlisp should be constructed to be portable, and the Interlisp support center should migrate this system to new hardware on a periodic basis.

10.3 PROGRESS

The project has completed its first phase with the delivery of Interlisp-VAX to sites running Berkeley UNIX. We are now engaged in the support of that product, its enhancement in both speed and functionality, and the development of secondary systems derived from the UNIX version of Interlisp-VAX.

As of June 1981, we had completed the design for Interlisp-VAX. ırtially completed coding of the virtual machine kernel and of support code written in LISP, and begun testing and debugging the code produced so far. The schedule we had established called for beta testing to start in January 1982 and a first release in March 1982. That schedule has been met in every respect.

The process of completing the coding and bringing up the Interlisp "environment" was very smooth, with no major unforeseen problems. We were running an essentially complete Interlisp "in house" by October 1981. We started beta testing in late November 1981, with both an expanded community at ISI and at selected test sites around the ARPA network.

As expected, numerous small bugs were shaken out during the test period, but the overall rate of problems was low enough to permit us to divert some attention from functionality to performance tuning. Among the improvements made after the test period began were large improvements in the efficiency of directory scanning and garbage collection, and a first-order speed up to free variable lookup.

In all, three or four rounds of testing were done on the UNIX version. In the course of testing, a number of large Interlisp systems were brought up on the VAX, none with any particular difficulty. Among them are Affirm, AP3, Hearsay, KL/One and ROSIE.

We also made a preliminary version of Interlisp to run under DEC's operating system for the VAX, VMS, using EUNICE, a UNIX compatibility package available from SRI. We did not progress far enough with this version to distribute it with the UNIX version, but we expect to do so soon. In February 1982 we "froze" development of Interlisp to prepare a stable version for release. Our official availability announcement was made in late February, and we have been distributing copies of Interlisp to UNIX sites since early March, 1982. We have distributed 91 copies of Interlisp to UNIX sites and 53 copies to VMS sites. An additional 54 requests are in progress.

We are currently working in three areas: First to improve the performance of Interlisp-VAX; second to increase sharing of sources with Interlisp-D, which we had temporarily diverged from; and third to complete, test, and release a version of Interlisp for VMS.

10.4 MILITARY IMPACT

The military relevance of this effort arises from military use of the many ARPA-sponsored research projects based on Interlisp.

This project is designed to fulfill the VAX requirements and to lay the foundation for the portability requirement.

10.5 CURRENT WORK

Our definite plans call for completion of the three current tasks, plus some new ones to be started when time allows. Our generally highest priority will be performance enhancements. We are not currently committed to any large development efforts, though several are possible.

Now that Interlisp-VAX has a user community, a significant amount of the staff's time will be spent on more or less routine but necessary tasks. In addition to bug fixing and the like, the most important task in this category is to continue to track the development of Interlisp-D; to keep the implementations as close as possible to the same. Toward that end, we are currently working to merge sources that diverged during the development, but which we have always intended should be shared.

Our first priority is the testing and delivery of the first release of Interlisp-VAX to sites running VMS. The adaptions necessary have already been coded and partially tested, but a lengthy shake down period is necessary to be sure that Interlisp, Eunice, and VMS are working together.

We hope to begin beta testing during June, but may be delayed as support for Eunice is not under our direct control. The test period will last at least one month before we ship copies to our waiting list of VMS sites.

Our second priority, though concurrent, is the second major release to UNIX sites, which, depending on the progress of VMS, may be concurrent with the first release to VMS sites. This release will include the result of greatly improved code produced by a new code generator, which is now being tested.

Our remaining priorities are split, but focused on improving performance of Interlisp as a component of a timesharing system rather than as a stand alone program. Some of this work will be dependent on progress by UC Berkeley toward the planned 4.2 release of UNIX.

Finally, on documentation, the immediate level of user documentation is adequate for the time being, but the formidable task of recreating and updating the Interlisp Manual is still outstanding. If that effort gets seriously started, we expect to participate. It is also essential that the internal documentation of our implementation, intended to benefit future implementors and maintainers, be completed before the memory fades.

11. STRATEGIC C3 SYSTEM EXPERIMENT SUPPORT

Research Staff:
Gary McGreal
Lynne Sims
Charlie Brown
Victor Ramos

11.1 INTRODUCTION

DARPA has defined an experiment in Strategic C3 systems to be conducted in cooperation with the WWMCCS System Engineer (WSE) and the Strategic Air Command (SAC). The concept of the experiment is to demonstrate and evaluate the use of new technologies (such as the ARPANET, packet radio, network security, and distributed knowledge-base techniques) for strategic command, control, and communication. Specific goals are to

- Demonstrate and evaluate the survivability of multinode computer-communication networks, including the ARPANET and packet radio, especially for remote access from both airborne platforms and surface sites.
- Explore replication and reconstitution of critical knowledge bases on this network in the face of a loss of a large number of links.
- Demonstrate and evaluate the rapid reconstitution of a network by rapid deployment of packet radio nets to reestablish connectivity between surviving elements of the network.
- Conduct experiments and exercises to evaluate the utility of such a network on a distributed knowledge base to support postattack C3 activities.

The DARPA experiment is defined as having three phases:

1. Phase I is planned to demonstrate air-to-surface packet radio links and gateways into the ARPANET as a first step in evaluating the feasibility of a truly survivable strategic network.
2. Phase II is directed toward creating a survivable message system and data bases through multiple copies of the critical components and data across the ARPANET.
3. Phase III will address the feasibility of rapid reconstitution of a strategic network by deployment of packet radio networks to reconnect surviving elements of the network.

11.2 PROBLEM BEING SOLVED

ISI is assisting in the development of a detailed technical plan for Phase II and providing the core facilities necessary to implement the plan. Specifically, SAC must have fairly broad-based experience with ARPANET-based on-line interactive computing prior to the introduction of Phase II. Installation of modems, installation of up to 30 interactive CRT terminals, user training, system software training and support, and on-site maintenance of equipment are part of the continuing program.

11.3 PROGRESS

During this period we have restructured the engineering support for the project. Fomerly, Charlie Brown was the only on-site representative at SAC from ISI. Programming and consulting support came from staff at ISI. Recently we hired Victor Ramos to take over the terminal maintenance support. Charlie Brown is now able to do more training and consulting on his own. This also allows for better coordination with the various DCS groups.

ISI contributed hardware and telecommunications coordination support for the Global Shield exercise. This proceeded smoothly in spite of limited time constraints.

A feasibility study was completed at the request of SAC AD to determine potential methods of providing ARPANET service to the third Air Division, Anderson AFB. A similar study is nearing completion to provide ARPANET connectivity to 7th Air Division at Ramstein AFB.

Maintenance, consulting and support for users and hardware was carried out at SAC, 15th AF, 8th AF and Peterson AFB. Currently 76 terminals are in use at SAC and 8 at the other sites.

11.4 FUTURE WORK

Although the major thrust of the DARPA Strategic C3 system experiment is well defined, the detailed definition of the Phase II program is still evolving. ISI will continue to assist DARPA in the shaping of this program, working to satisfy the communication and computer resource requirements of SAC headquarters. In particular, ISI will do the following:

- Continue to provide on-site maintenance support for the required equipment.
- Continue to plan and assist in implementing improved SAC connectivity to the ARPANET.
- Install and maintain terminals and communication equipment for connection of two numbered Air Force bases to the ARPANET for their participation in the experiment.
- Continue to supply programming and engineering support for Phase II of the experiment.

We will provide two on-site technicians at SAC who will be responsible for the identification of system malfunctions and primary maintenance of on-site equipment. They will be supplied with appropriate spare parts and will have maintenance contracts with the equipment vendors. Further support will be available from ISI in terms of additional spare parts, systems expertise, and documentation as required. The on-site maintenance technicians will also be responsible for the off-site terminals at Peterson AFB, Barksdale AFB, March AFB, and any new locations. The senior technician will coordinate requests of SAC AD with SRI and XPFC under the supervision of ISI management. The senior technician will provide training and consulting with backup from ISI as required.

As the Phase II program evolves, ISI will assist DARPA as required to shape this effort. We will continue to investigate the data requirements for SAC Headquarters, the Headquarters Emergency Relocation Team, and the ACCESS system. As specific research tasks are defined ISI may choose to submit new proposals to address one or more of these tasks.

12. NEW COMPUTING ENVIRONMENT

Research Staff:
Joel Goldberger
Jim Koda
Craig Rogers

12.1 PROBLEM BEING SOLVED

The NCE project goal is to adapt developing computer technologies to serve the research and military user communities. The resulting model computing environment will serve four purposes:

1. It will provide a very significant improvement in languages, system support, and additional computing cycles to the ongoing ISI research effort.
2. It will serve as a testbed and eventual existence proof for the implemented technology.
3. It will serve as a proving ground for local computer environments targeted for ARPA and the military community, as well as a device for trying out new research ideas that will eventually be adapted by those communities.
4. The small size, portability, and local computing capability of PCs will allow for experimentation and realization of command and control requirements for an environment that can exist in mobile command centers, be they vans, ships, or airplane based.

12.2 BACKGROUND

For convenience in formulating a generic model of a computing environment, computational activity can be split into two major types: interactive computing and CPU bound computing. Our model in this proposal assumes that the user will have at least the computing capacity in his personal computer (PC) to allow him to do all interactive computing locally. If the PC is fast enough it may be able to do the CPU bound computing as well. If the frontend PC is not fast enough, then there is a requirement for more powerful rear-end servers. All scenarios for a personal computing environment are some combination of PCs and servers.

Generally, interactive computing will be considered to include editing, mail handling, administrative tasks (interactive calculations, spread sheets, calendar maintenance), directory maintenance, conferencing, small program generation/debugging, and system status checking.

CPU bound computing includes LISP programming, large program compiles, large program execution, large file transfer, sorts, searching, and database management.

The guiding principle in the design of this environment will be full utilization of current hardware technology as it is represented by personal computers, bit mapped displays, and pointing devices (mice). A common failure in the utilization of new hardware technology is to drag along the prejudices and limitations of operating system software that was suitable for an earlier hardware base. Current applications software rarely takes advantage of what are the now commonplace features of smart terminals (blinking characters, highlighting, reverse video, etc.). An environment that further

perpetuates such a teletype oriented user interface onto bitmapped displays is an unacceptable waste.

Human engineering is an area that has been frequently overlooked in computer system implementations. Historically, operating systems have evolved from a level of low complexity/functionality to high complexity/functionality not out of a coordinated plan, but rather as the result of the programming requirements of individual users. Functionality has been tacked on, rather than integrated into systems.

Computers are a tool to aid in the solving of problems. Time wasted dealing with a poorly implemented system detracts from the potential benefits. Hidden costs of poor human engineering can result from time wasted learning non-mnemonic commands, time wasted deciphering bad documentation, time spent tracking down errors, work lost through system malfunction or inadvertent deletion, time spent training new users and the cost of staff who should but don't use the computers because of the difficulties resulting from bad design.

Users who are already trained on one computer system are frequently frustrated by the design flaws of another. To be successful NCE must offer a large tangible improvement over existing systems (particularly TOPS-20) and at the same time minimize retraining of users familiar with other environments.

The NCE will have to perform a set of functions in support of its users. The following functions are included

- Word Processing Support
- Network Connectivity
- Electronic Mail Services
- Simple Programming Capability
- Large Program Support
- Graphics Services
- Administrative Support Software
- File Server and Backup Support
- System Support

A brief description of each functional area is presented in the following paragraphs.

Word Processing Support. Currently the ISI community spends a great deal of time manipulating text and program source files. The current environment has failed to integrate the various components of this activity. Much processor time and system resources are wasted. The generation of research reports, documents for publication, or documents for presentation at conferences generally involves the use of a document compiler (SCRIBE, RUNOFF, or PUB). These programs, which are separate from the editors, provide formatting and font support for the document. Neither the output of the editor nor that of the document compiler ever lets the user see what his final document will look like when printed on the laser printer. Generally this results in the user going through several iterations of a document before it is formatted as was intended. Bit mapped displays offer an escape from this problem. With a properly sized display and sufficient resolution, font information can be embedded in the user's original file. The editor software should be able to interpret the embedded positioning and font codes and display a reasonable representation of what

the document will look like when printed on a high resolution laser printer. The NCE editor is expected to make full use of the available hardware technology. The commands interface should be oriented towards the bit mapped display and allow full utilization of the pointing device and its buttons. Help and command facilities should be able to work in a separate window without interfering with the text. A menu oriented command interface should be an available mode. The display should be large enough to hold an almost full sized facsimile of the document while allowing sufficient extra window space for help text, macro definitions, and command data. The Xerox Star display which is 10 and 1/2 by 13 and 1/2 inches with a resolution of 72 dots per inch is representative of an appropriate display. Editor functionality should include (but not be limited to) the capabilities described in Appendix III of Teresa Roberts' "Evaluation of Text Editors" (Xerox PARC, 1979). While editing is generally thought of as an interactive function, certain aspects of editing are CPU bound. Such common activities as string searches and justification fall into this category. If the local processor lacks the capability of processing these functions rapidly, then a remote server should be available to handle them.

Network Connectivity. The NCE local network needs to be connected to the ARPANET and other networks (TELENET, TYMNET, internal research nets) as required by individual research projects. The local network protocols will be based on the DoD standard TCP/IP. Users must be able to gain access to the local net and use services and attach to workstations via the ARPANET. Similarly, users need to be able to connect to resources via direct dial telephone lines. The network architecture should consider the constraints and requirements that may be associated with operating separate processors on the Defense Digital Network (DDN). Gateways to research subnets and experimental hardware should be designed so that problems with the research hardware or software do not impact the operational facility. Users at any workstation should be able to connect to one or more other workstations for purposes of conferencing. Users at any workstation should be able to connect to backend servers and receive compacted data representations of text, font information, windowing information, and graphic representations in raster op format. Multiple local nets should be implemented as required to sustain bandwidth required for an acceptable service level. Users of the local net should not be in a position where they are competing for resources in a way that affects interactive processing.

Electronic Mail Services. The electronic mail services currently available provide critical services to the research community. Messages are used for communication with other researchers over the ARPANET, with ARPA program managers, and for local communications and project control. The functionality of the larger mail handlers (Hermes or MM) meet most of the needs of the current environment. It is expected that the NCE mail services will provide similar service. All bit map editing functions should be available as part of the mail program. Users should be able to select and read messages through the use of the pointing device and menu oriented selection procedures. It should be possible to imbed font and graphic data in messages that are to be distributed locally or to other sites with similar bit mapped displays and hardcopy devices. Mail distribution and name server services should be provided centrally. A centralized capability to repack messages for transmittal over networks with different protocols may be useful. A store and forward capability for utilization of phonelines and late night DDD rates could be useful for making connections to sites lacking direct hardwire connections.

Simple Programming Capability. Regardless of whether we choose a relatively low powered workstation that requires substantial support from backend servers, or a medium powered station where most work is done locally, it will be necessary to provide access to an easy to use block structured programming language with an efficient compiler. This language will be used for small

program implementations and algorithm checking. A compatible version of the language should exist on a server for larger programs and production software development. While the choice of this language should be made carefully, viable possibilities include Pascal, C, Ada, Mainsail, SmallTalk, or Mesa. Other candidates may be more or less suitable depending upon the hardware environment and operating system chosen for the NCE. Ideally, the debugging facility should be integrated into the editor and allow source code breakpointing, editing, and simultaneous program execution using multiple windows. Such a system now exists in support of the Mainsail environment. Mainsail itself may be a poor choice for other reasons.

Large Program Support. Workstations with limited local computing power will need to be supplemented by a variety of servers providing higher bandwidth computing. LISP applications are the most obvious example. Similar servers may be necessary for the block structured language, database activities, and other CPU bound functions (sorting, searching, array processing, etc.). Communications between the server and the workstation will be through the local network graphics protocol.

Graphics Services. Graphic and windowing primitives should be implemented at a low level and be available for user programs. Support should be available for the various fonts that are available on the hardcopy device. The support for the pointer device should also be implemented at a low level and be available for adaptation by user programs. A graphics based user program interface should exist that allows command input from the pointer device, understands windows, and generates selection menus from tables in the program. This is envisioned as an upgrade of TOPS-20's COMND JSYS monitor call that understands bit mapped displays and pointers. Basic routines for describing window location, objects (or characters), line drawing, curve drawing, and object vectoring should be available in the workstation.

Administrative Support Software. This class of software includes interactive calculators, spread sheet programs (Visicalc), directory maintenance software, system status programs, and calendar software. It will be a goal of NCE to integrate this software thoroughly into the bit map/pointer hardware environment.

Fileserver and Backup Support. The services, security, and functionality of the LOCUS fileserver, a system being developed at UCLA, are deemed to serve the needs of the user community. The file system chosen to implement the LOCUS concepts should be as robust as possible. A TOPS-20 file system is probably sufficient, while UNIX is not. It is likely that some other modern file system will be chosen (such as VAX/VMS). The known deficiencies of the UNIX filesystem (simpleminded device drivers, poor recovery tools, lack of dynamic BATBLK allocation, inability to increase a filesystem size dynamically) make a UNIX based fileserver a poor choice. The fileserver should have tape drives to back up the disk. The required functions are incremental dumps, full dumps, restores, archive and migration.

System Support. System support services should be available for the NCE as they are on TOPS-20. User services should be available to handle problem reports, maintain fileserver directories, load balancing, network balancing, documentation, and training. The operations staff should be available to ensure operational status of the system and its components, sort output, and to convey problems to the hardware and software groups. Software support should be available to maintain the NCE developed software and to install new software as the technology and people's needs change. The hardware should be maintainable locally. Backup workstations should be available to replace broken units while they are being repaired. No singlepoint failures should cause lack of access to the whole facility.

NCE Performance. One of the primary concerns of the NCE project is system performance. One of the main problems with the current TOPS-20 system is irregularity of service and lack of peak load computing power. Users should not be impacted by the load requirements of other users. The basic required benchmark is that the cumulative computing capacity results in an effective service level to the user of ten times the current service level during peak load. System availability is also an issue. The system should be up and available to individual users at least 99% of the time.

12.3 PROGRESS

To date a variety of possible hardware bases have been analyzed to determine their suitability to our evaluation criteria. Processors that have been reviewed include VAX 11/730, Perqs, Apollos, Wicats, Suns, Xerox 1100s, and Symbolics 3600s.

We have decided the optimum processor for our needs is the Xerox Dandelion running with the Mesa programming environment. This processor offers a combination of high speed, low cost, sophisticated software, and integrated printing peripherals that is unmatched by the competition.

Since our needs require detailed access to the Xerox software and to the Mesa programming environment (not an official Xerox product at this time) we have been negotiating extensively with Xerox to meet our requirements.

12.4 MILITARY IMPACT

The possibility of high powered network based processors allows for sophisticated solutions to command and control problems. Survivable networks coupled with reconstitutable databases accessible from mobile packet radio units allow for effective communication during trans- and post-attack periods. Such systems will allow the surviving command structure to recover returning missions, evaluate surviving resources, and retarget those resources in an extended nuclear confrontation.

The effectiveness of distributed command and control computers will depend upon the tools and user interfaces available on them as well as on their utilization during peace time (if not used on a day by day basis such sophisticated tools are not as effective during the specialized scenarios for which they are designed because of the need for capable operating personnel).

12.5 FUTURE WORK

During FY83 we expect to acquire a limited number of PCs in order to begin developing the software they need to be useful research and development tools. In particular PCs will need to be integrated into the ARPANET environment with its mail programs and particular protocols. In addition we will be developing systems to enable dissimilar graphic workstations to communicate using bitmap protocols. This will enable users to benefit from software that needs to run on high speed and/or specialized processors.

The eventual goal is to have a workstation environment that allows the research and administrative programming that goes on at ISI to be carried out on the PCs. At this point the system and its

programs will be available for porting to military environments. It will arrive as a well tested and reliable environment.

13. COMPUTER RESEARCH SUPPORT

Director: Daniel C. Lynch
Deputy Director: Gary L. McGreal
Technical Assistant: Dale Russell
Manager NOSC Facility: Serge Polevitsky

Technical Staff:

Systems Software:
Dennis Smith
David Bilkis
Helaine Fehling
David Kompel
Ellen Lohneiss
William Moore
Craig Rogers

Hardware:
Daniel Pederson
Glen Gauthier
Ramon Gonzales
Norman Jalbert
Raymond Mason
Daniel Trentham
Leo Yamanaka

Operations:
James Hurd
Ron Shestokes
Steve Carroll
Sam Delatorre
Walt Edmison
Brent Gesch
Roylene Jones
Joseph Kemp
Patrick Saunders
Tom Smith
Toby Stanley
Mike Zonfrillo

Network Services:
Lynne Sims
Craig Ward
Vicki Gordon
Linda Sato
Robin Swick
Wayne Tanner
Debbie Williams

Support Staff:
Sara Tietz
Denise DeGemmis
Manon Levenberg

13.1 PROBLEMS BEING SOLVED

The TENEX/TOPS-20 project is responsible for providing reliable computing facilities on a 24-hour, 7-day schedule to the ARPANET research and development community. At the same time, the project makes available to ARPANET users the latest upgrades and revisions of hardware and software. The project provides continuous computer center supervision and operation, and a full-time customer-service staff that is responsive to user inquiries. This project supports two computer installations: the larger at ISI's main facility in Marina del Rey, the smaller at the Naval Ocean Systems Center (NOSC) in San Diego.

13.2 GOALS AND APPROACHES

The TENEX/TOPS-20 project provides support in four interrelated though distinct areas: Hardware, System Software, Operations, and Network Services. The goals and approaches of each are summarized below.

Hardware

To achieve a reliability goal of 98.7 percent scheduled uptime, the preventive and remedial maintenance responsibilities have been assigned to an in-house computer maintenance group. This group provides 20-hour, 5-day on-site coverage and an on-call standby coverage for after hours. The maintenance philosophy of this group can best be stated as "if it isn't broken, don't fix it." For this approach to be successful, preventive maintenance is very closely controlled, and on-line diagnostics and analysis are emphasized. A primary component in the reliability and availability of the hardware is the physical environment in which it exists. Accordingly, a significant amount of time and resources is expended in insuring that the best, most cost-effective environmental controls are used at both facilities that ISI services.

System Software

The software group's primary goal is to install and maintain, at maximum reliability, ISI's TENEX and TOPS-20 operating systems and applications software. In order to accomplish this goal, the group provides 24-hour, 7-day coverage to analyze system crashes and to provide appropriate fixes. In addition, it is the group's responsibility to install, debug, and modify both the latest monitor versions and the associated subsystems available from the vendor.

Operations

The operations staff is responsible for operating the computers and watching over the systems and the environment. At the Marina del Rey facility there is 24-hour, 7-day on-site coverage. At NOSC normally there is coverage from 7:00 am to 4:00 pm daily; special exercises sometimes require extended hours. When a problem occurs, the on-duty staff isolates it and takes appropriate action. On the night and weekend shifts, the operations staff respond directly to user inquiries. Proper training, experience, and familiarity with the environment are especially important.

Network Services

The customer service group, based in Marina del Rey, provides a two-way communication link between the users and consulting staff. This is accomplished by maintaining a 12-hour, 5-day on-duty staff for prompt problem resolution and rapid information exchange, both on-line and by telephone. The group offers introductory training in the use of both the hardware and software tools available on the ISI TENEX/TOPS-20 systems as well as documentation for new users of the ARPANET, and it also assists in the formulation of user training experiments for large, ARPANET-based military experiments, for example, the U.S. Army XVIII Airborne Division Army Data Distribution System at Ft. Bragg, N.C., and the C3/ARPANET Experiment at Headquarters Strategic Air Command at Offutt AFB, Nebr. Appropriate documentation is constantly being generated and distributed to individual users, as well as to remote user group liaison personnel; this documentation ranges from that for the novice to that for more experienced users. In accordance with IPTO guidelines, the customer-service group provides appropriate accounting data to ARPA.

13.3 PROGRESS

In the past year, ISI has significantly improved in its ability to meet the increasing demand for TENEX/TOPS-20 computer cycles, by users on the ARPANET and at ISI. These improvements fall into the following catagories: Environmental Changes, Hardware Additions and Software Enhancements.

Environmental Changes

Because the reliability of the computer systems was being degraded by environmental problems, specifically, power surges and inadequate air conditioning, ISI upgraded both the power sources and the air-conditioning facilities. Raised floor, conditioned power, new air conditioning, power distribution units, halon fire protection units, CCTV security monitors and increased floor space were added to the facility.

Hardware Additions

In addition to the environmental enhancements descibed above a number of hardware additions were made in FY82. These include a Micom terminal switch, VAX 11/780, 10 VAX 11/750s, and a variety of research processors. The current local hardware configuration is shown in Figure 13-1. The remote NOSC hardware configuration is shown in Figure 13-2.

System Software Enhancements

During FY82 the major effort centered upon preparation of the TOPS-20 systems for the conversion from NCP network protocols to the new DoD standard TCP/IP suite of protocols. This work is a joint effort of DEC, BBN, and ISI. ISI has been developing the applications level software (SMTP mail telnet, FTP, etc.) while the other contractors have concentrated on the required OS development. In addition to the TCP effort a number of software additions and enhancements have been made to the DEC TOPS-20 systems. These include NCP Calc, a Visicalc look-a-like program; new versions of the MM mail program, a calendar program, and various modifications to the subsys programs.

During this period we installed our first VAX running under the VMS operating system. This involved generating custom network software, and installing user support programs (Mail, Editors, Network Applications and Print spoolers.)

13.4 MILITARY IMPACT

ISI's computer centers provide ARPANET cycles 24 hours a day, 7 days a week to the Strategic Air Command, Naval Ocean Systems Center, and Ft. Bragg. In addition to supplying machine time, this project has provided additional support in the following areas:

- Training, documentation, and modifications as requested by user groups for NLS.
- Planning support and training in ISI systems software for the installation of an on-site DEC System 2060 at Gunter AFB.
- Support for the production of AFM 67-1 with NLS.

13.5 FUTURE WORK

The Computer Research Support project will continue to provide computing service to the ARPA research community, provide and support software packages for the ARPANET community, and offer a program of technology transfer through the customer-service group.

The planned program for 1983 includes a number of major changes. TCP/IP is scheduled for implementation on Jan. 1, 1983. The first monitor release will be followed shortly by a version based

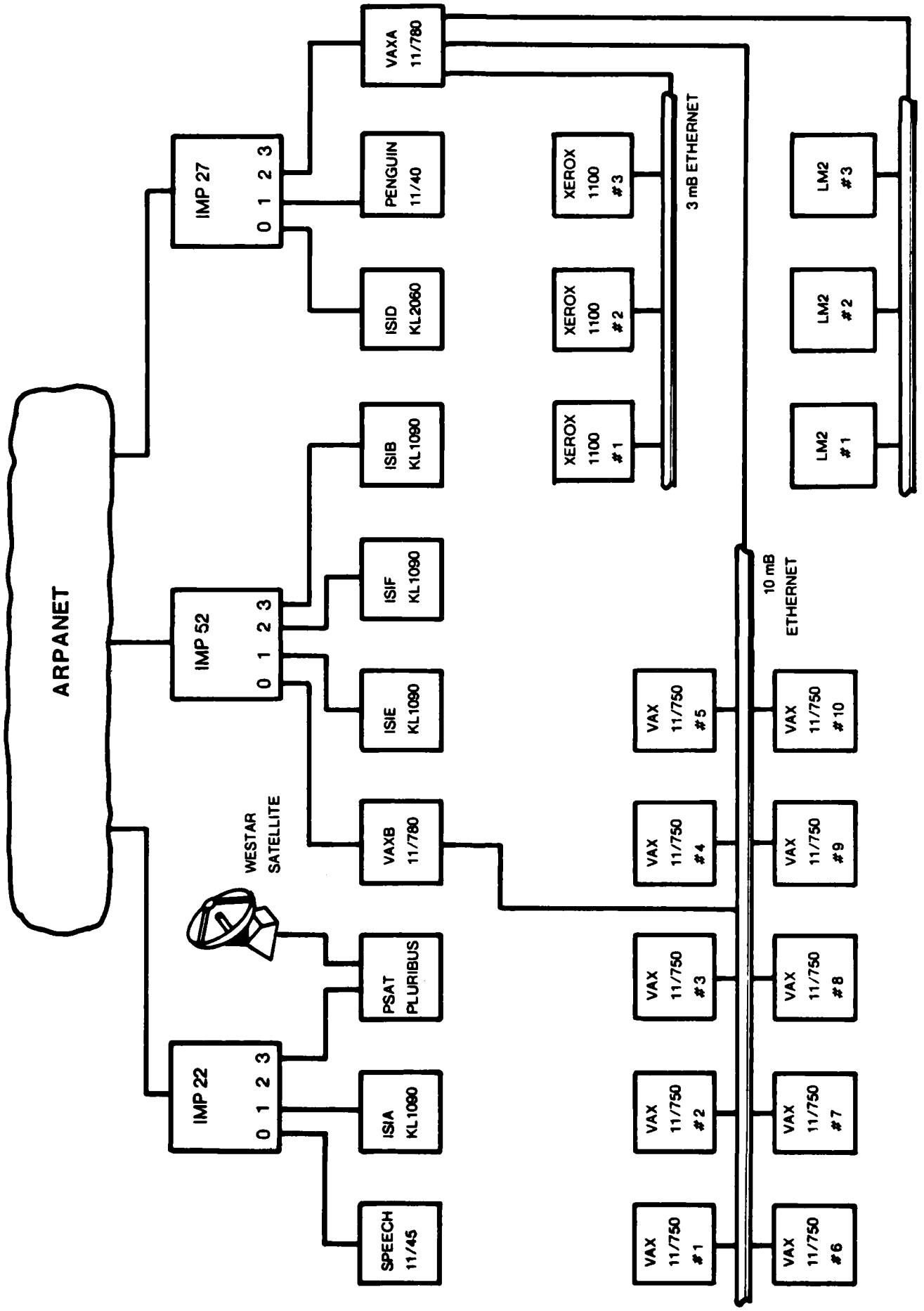


Figure 13-1: Diagram of local ISI ARPANET facility

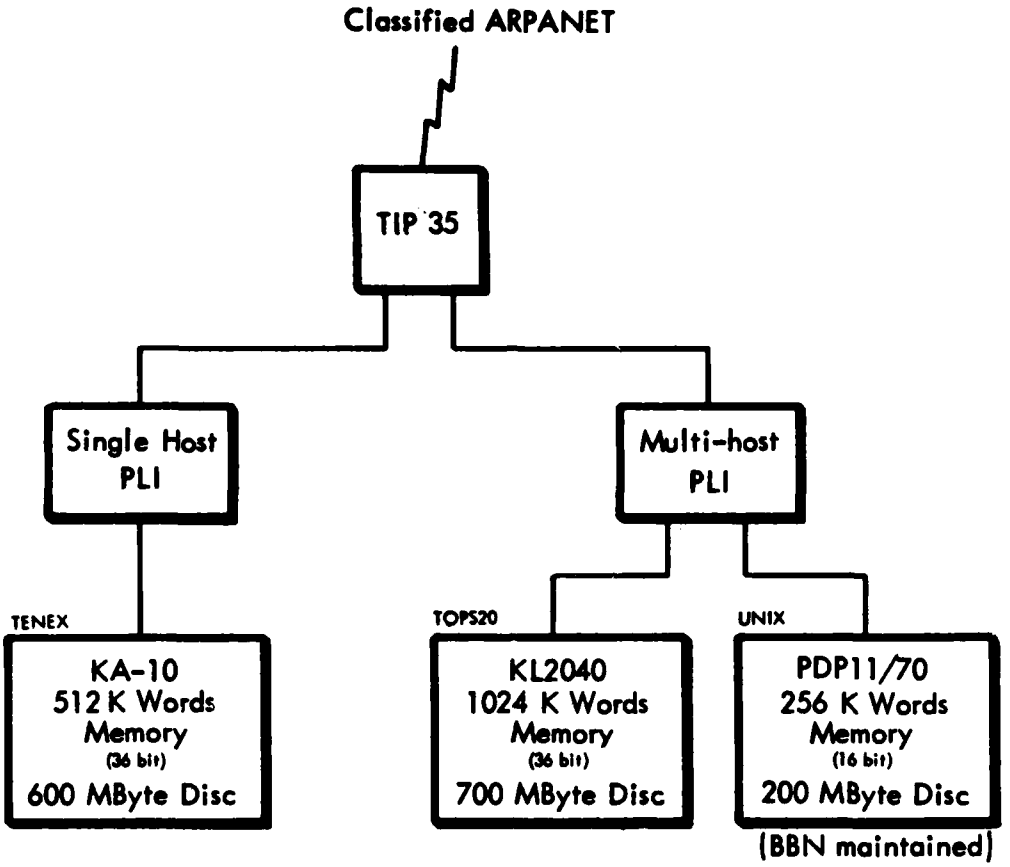


Figure 13-2: Diagram of remote ISI ARPANET facility at NOSC

on the DEC software. This version will more closely tie the network activities to the internal concepts of files and devices already in TOPS-20. This should improve efficiency and maintainability.

New releases of TOPS-20 (version 5.0), VMS (version 3.0), and Berkeley UNIX are expected in FY83 and will be installed on the appropriate machines. In 1983 ISI will take major steps toward moving its research program from timesharing computers to network based PCs.

14. PUBLICATIONS

Balzer, R., Don Cohen, M. S. Feather, N. Goldman, W. Swartout, and D. S. Wile, "Operational specification as the basis for specification validation," in *II Software Technology Seminar, "Software Factory Experiences,"* Capri, Italy, May 1982. (Soon to be published in North-Holland Proceedings, *Theory and Practice of Software Technology*.)

Balzer, R., D. Dyer, M. Fehling, and S. Saunders, "Specification-based computing environments," in *Proceedings of the Eighth International Conference on Very Large Data Bases*, Mexico City, September 1982.

Balzer, R., N. Goldman, and D. S. Wile, "Operational specification as the basis for rapid prototyping," in *Pre-Proceedings of the ACM SIGSOFT Software Engineering Symposium on Rapid Prototyping*, Columbia, Maryland, April 1982. (Also to appear in the published proceedings of the conference.)

Barnett, J. A., "Computational methods for a mathematical theory of evidence," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 868-875, Vancouver, B. C., August 1981.

Barnett, J. A., "Some issues of control in expert systems," in *Proceedings of the IEEE Conference on Cybernetics and Society*, pp. 1-5, University of Washington, Seattle, October 1982.

Bates, R. L., D. Dyer, and J. A. G. M. Koomen, "Implementation of Interlisp on the VAX," in *Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, pp. 81-87, August 1982.

Berthomieu, B., *Algebraic Specification of Communication Protocols*, USC/Information Sciences Institute, RR-81-98, December 1981.

Cohen, Danny, "A message base user interface," in *Proceedings of INFOCOM-82*, pp. 161-162, Las Vegas, March 1982.

Cohen, Danny, "A voice message system," in R. Uhlig (ed.), *Computer Message Systems*, pp. 17-28, North-Holland, 1981.

Cohen, Danny, "DATACOM - The modern way," *Datamation* 27, (8), August 1981, 177-178.

Cohen, Danny, "Internet mail forwarding," in *Proceedings Compcon Spring 82*, pp. 384-390, San Francisco, February 1982. (Also to appear in *Proceedings of the Fourth International Workshop of IFIP--WG.6.5*, Rocquencourt, France, April 1982.)

Cohen, Danny, "On holy wars and a plea for peace," *IEEE Computer* 14, (10), October 1981, 48-54.

Cohen, Danny, "The impact of VLSI on peripheral array processors," in *Proceedings of the Conference on Peripheral Array Processors*, pp. 33-38, San Diego, October 1982, *Simulation Series* Vol. 11, No. 2.

Cohen, Danny, "The servant's dilemma," *Datamation* 27, (12), November 1981, 241.

Cohen, Danny, "The VLSI approach to computational complexity (by Professor J. Finnegan)," in H. T. Kung, R. Sproull, and G. Steele (eds.), *VLSI Systems and Computations, Proceedings of the CMU Conference*, pp. 124-125, Computer Science Press, October 1981.

Cohen, Danny, "Using local area networks for carrying online voice," in P. C. Ravasio, G. Hopkins, and N. Naffah (eds.), *Local Computer Networks*, North-Holland, 1982, pp. 13-21. (Also appears in *Proceedings of the IFIP-TC6 International In-Depth Symposium on Local Computer Networks*, Florence, Italy, April 1982.)

Cohen, Danny, and L. Johnsson, "A formal derivation of array implementation of FFT algorithms," in *Proceedings of the USC Workshop on VLSI and Modern Signal Processing*, pp. 53-63, September 1982. (Also appears as California Institute of Technology Computer Science report 5043:TM:82.)

Cohen, Danny, and L. Johnsson, "Algebraic description of array implementation of FFT algorithms," in *Proceedings of the Twentieth Annual Allerton Conference on Communication, Control, and Computing*, University of Illinois at Urbana-Champaign, October 1982.

Cohen, Danny, and L. Johnsson, "A mathematical approach to modelling the flow of data and control in computational networks," in H. T. Kung, R. Sproull, and G. Steele (eds.), *VLSI Systems and Computations, Proceedings of the CMU Conference*, pp. 213-225, Computer Science Press, October 1981.

Cohen, Danny, and L. Johnsson, "A mathematical approach to the design of VLSI networks for real-time computation problems," in *Proceedings of the Real-Time Systems Symposium*, pp. 32-40, IEEE, 1981.

Cohen, Danny, and L. Johnsson, "A VLSI approach to real-time computation problems," in *Proceedings of the Society of Photo-Optical Instrumentation Engineers: Real Time Signal Processing*, August 1981.

Cohen, Danny, and L. Johnsson, "The impact of VLSI on signal processing," in *Proceedings of the USC Workshop on VLSI and Modern Signal Processing*, pp. 153-156, September 1982.

Cohen, Danny, and J. Moore, "Assessing the priority schemes in two verbal information-delivery systems," in *Standardization for Speech I/O Technology*, National Bureau of Standards, March 1982.

Cohen, Danny, and J. Moore, "What do you say after you say *Pull up and go around!*? (The problem of talking to a pilot)," in *Proceedings of the Conference on Simulation for Aircraft Test and Evaluation*, pp. 128-131, Navy Air Test Center, Patuxent River, Maryland, March 1982.

Cohen, Danny, and V. Tyree, "Quality control from the silicon broker's perspective," *VLSI Design* 3, (4), July/August 1982, 24-30.

Cohen, Don, W. Swartout, and R. Balzer, "Using symbolic execution to characterize behavior," in *Pre-Proceedings of the ACM SIGSOFT Software Engineering Symposium on Rapid Prototyping*, Columbia, Maryland, April 1982. (Also to appear in the published proceedings of the conference.)

Cole, E. R., "Packet voice: When it makes sense," *Speech Technology* I, (3), September/October 1982, 52-61.

Cole, E. R., "PVP - A Packet Video Protocol," USC/Information Sciences Institute, W-Note 28, August 1981.

Erman, L. D., F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty," in B. L. Webber and N. Nilsson (eds.), *Readings in Artificial Intelligence*, pp. 349-389, Tioga, Palo Alto, Calif., 1981.

Erman, L. D., P. E. London, and S. F. Fickas, "The design and an example use of Hearsay-III," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 409-415, Vancouver, B. C., August 1981.

Feather, M. S., "A system for assisting program transformation," *ACM Transactions on Programming Languages and Systems* 4, (1), January 1982, 1-20.

Feather, M. S., "Mappings for rapid prototyping," in *Pre-Proceedings of the ACM SIGSOFT Software Engineering Symposium on Rapid Prototyping*, Columbia, Maryland, April 1982. (Also to appear in the published proceedings of the conference.)

Feather, M. S., "Program specification applied to a text-formatter," *IEEE Transactions on Software Engineering* SE-8, (5), September 1982, 490-498.

Finn, G., and J. Postel, "Data structure and presentation control for multimedia computer mail," in *Proceedings EASTCON*, November 1981.

Gallenson, L., and J. Goldberg, "QPRIM System: Tool Builders' Manual," USC/Information Sciences Institute, WP-20, 1981.

Hayes-Roth, F., P. Klahr, and J. Mostow, "Advice taking and knowledge refinement: An iterative view of skill acquisition," in J. A. Anderson (ed.), *Cognitive Skills and Their Acquisition*, pp. 213-253, Erlbaum, 1981.

Karplus, W., and Danny Cohen, "Architectural and software issues in the design and application of peripheral array processors," *IEEE Computer* 14, (9), September 1981, 11-17.

Katz, A., and J. Postel, "Decoding Facsimile Data from the Rapicom 450," USC/Information Sciences Institute, RFC 798, September 1981.

Katz, A., and J. Postel, "Format for Bitmap Files," USC/Information Sciences Institute, RFC 797, September 1981.

Kini, V., D. Martin, and A. Stoughton, "Testing the INRIA Ada formal semantic definition: The USC-ISI formal semantics project," in *AdaTec Conference Proceedings*, pp. 120-128, Association for Computing Machinery, October 1982.

Marcus, L., *On the Equivalence of If-Then-Else and Max-Min*, USC/Information Sciences Institute, RR-81-92, August 1982.

Marcus, L., *State Deltas that Remember: A Formalism for Describing State Changes*, USC/Information Sciences Institute, RR-81-93, May 1982.

Mark, W., "Representation and inference in the Consul system," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 375-381, Vancouver, B. C., August 1981.

Merritt, I. H., and R. H. Parker, "Switched Telephone Network Interface Card (STNI)," November 1981.

Mockapetris, P. V., *Communication Environments for Local Networks*, USC/Information Sciences Institute, RR-82-103, December 1982.

Moses, L., *An Introductory Guide to TOPS-20*, USC/Information Sciences Institute, TM-82-22, June 1982.

Mostow, J., "Learning by being told: Machine transformation of advice into a heuristic search procedure," in R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning*, Tioga, Palo Alto, 1982.

Mostow, J., and M. Lam, "Transformational VLSI Design: A Progress Report," 1982. (Written version of a talk presented at the 1982 IEEE Workshop on VLSI and Software Engineering.)

Postel, J., "Address Mappings," USC/Information Sciences Institute, RFC 796, September 1981.

Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC 790, September 1981.

Postel, J., "Clarification of TCP end of letter," *ACM Computer Communication Review* 11, (13), July 1981, 9.

Postel, J., "Computer Mail Meeting Notes," USC/Information Sciences Institute, RFC 805, February 1982.

Postel, J., "Internet and Transmission Control Protocol Specifications," USC/Information Sciences Institute, RFCs 790-796, February 1982.

Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," USC/Information Sciences Institute, RFC 792, September 1981.

Postel, J., "Internet Protocol - DARPA Internet Program Protocol Specification," USC/Information Sciences Institute, RFC 791, September 1981.

Postel, J., "Multimedia Mail Meeting Notes," USC/Information Sciences Institute, RFC 807, February 1982.

Postel, J., "NCP/TCP Transition Plan," USC/Information Sciences Institute, RFC 801, November 1981.

Postel, J., "Request for Comments on Requests for Comments," USC/Information Sciences Institute, RFC 825, November 1982.

Postel, J., "Service Mappings," USC/Information Sciences Institute, RFC 795, September 1981.

Postel, J., "Simple Mail Transfer Protocol," USC/Information Sciences Institute, RFC 788, November 1981.

Postel, J., "Simple Mail Transfer Protocol," USC/Information Sciences Institute, RFC 821, August 1982.

Postel, J., "Summary of a Computer Mail Service Meeting Held at BBN on 10 January 1979," USC/Information Sciences Institute, RFC 808, February 1982.

Postel, J., "The Remote Telnet User Telnet Service," USC/Information Sciences Institute, RFC 818, November 1982.

Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification," USC/Information Sciences Institute, RFC 793, September 1981.

Postel, J., C. Sunshine, and Danny Cohen, "The ARPA Internet protocol," *Computer Networks* 5, (4), July 1981, 261-271.

Postel, J., C. Sunshine, and Danny Cohen, "Recent developments in the DARPA Internet program," in *Pathways to the Information Society: Proceedings of the Sixth International Conference on Computer Communication*, ICCC, London, September 1982.

Postel, J., and J. Vernon, "Requests for Comments Summary - Notes 600-699," USC/Information Sciences Institute, RFC 699, November 1982.

Postel, J., and J. Vernon, "Requests for Comments Summary - Notes 700-799," USC/Information Sciences Institute, RFC 800, November 1982.

Schwabe, D., "Formal specification and verification of a connection-establishment protocol," in *Proceedings of the Seventh Data Communications Symposium*, pp. 11-26, Mexico City, October 1981. (Also appeared as USC/Information Sciences Institute, RR-81-91, January 1981.)

Shoch, J. F., Danny Cohen, and E. A. Taft, "Mutual encapsulation of internetwork protocols," *Computer Networks* 5, (4), July 1981, 287-300.

Sluizer, S., and J. Postel, "Mail Transfer Protocol: ISI TOPS20 Implementation," USC/Information Sciences Institute, RFC 784, July 1981.

Sluizer, S., and J. Postel, "Mail Transfer Protocol: ISI TOPS20 MTP-NIMAIL Interface," USC/Information Sciences Institute, RFC 786, July 1981.

Sluizer, S., and J. Postel, "Mail Transfer Protocol: ISI TOPS20 File Definitions," USC/Information Sciences Institute, RFC 785, July 1981.

Smallberg, D., "Who Talks TCP? - Survey of 7-Dec-82," USC/Information Sciences Institute, RFC 832, December 1982.

Smallberg, D., "Who Talks TCP? - Survey of 14-Dec-82," USC/Information Sciences Institute, RFC 833, December 1982.

Smallberg, D., "Who Talks TCP? - Survey of 22-Dec-82," USC/Information Sciences Institute, RFC 834, December 1982.

Smallberg, D., "Who Talks TCP? - Survey of 28-Dec-82," USC/Information Sciences Institute, RFC 835, December 1982.

Stefik, M., J. Aikins, R. Balzer, et al., *The Organization of Expert Systems: A Prescriptive Tutorial*, Xerox Palo Alto Research Center, Report No. VLSI-82-1, January 1982.

Su, Z., and J. Postel, "The Domain Naming Convention for Internet User Applications," Network Information Center, SRI International, RFC 819, August 1982.

Sunshine, C. (ed.), *Communication Protocol Modeling*, ARTECH House, Dedham, Mass., 1981.

Sunshine, C. (ed.), *Proceedings of the Second International Workshop on Protocol Specification, Testing, and Verification*, North-Holland, 1982.

Sunshine, C., "Addressing problems in multi-network systems," in *Proceedings of INFOCOM-82*, Las Vegas, Nevada, March 1982.

Sunshine, C., "Comments on the NBS Transport Protocol Proposal," USC/Information Sciences Institute, IEN 195, August 1981.

Sunshine, C., "Formal modeling of communication protocols," in S. Schoemaker (ed.), *Computer Networks and Simulation*, North-Holland, 1982 (revised edition). (An earlier version of this paper appeared as USC/Information Sciences Institute, RR-81-89, March 1981.)

Sunshine, C., "Guest editorial: Protocol specification, testing, and verification," *Computer Networks* 6, (6), December 1982, 375-376.

Sunshine, C., "Protocol Specification and Verification Work at USC/ISI: Summary Report," USC/Information Sciences Institute, IEN 211, August 1982.

Sunshine, C., "Protocol workshop report," *ACM Computer Communication Review* 11, (3), July 1981, 7-8. (Also appears in *Computer Networks* 5, (4), July 1981, 313-314.)

Sunshine, C., "Routing for Stub Gateways," USC/Information Sciences Institute, June 1982.

Sunshine, C., "Transport protocols for computer networks," in F. Kuo (ed.), *Protocols and Techniques for Data Communication Networks*, pp. 35-37, Prentice-Hall, 1981.

Sunshine, C., and D. Smallberg, *Automated Protocol Verification*, USC/Information Sciences Institute, RR-83-110, October 1982.

Sunshine, C., Danny Cohen, and J. Postel, "Comments on Rosen's Memos (IENs 184, 187-189)," USC/Information Sciences Institute, IEN 191, July 1981.

Swartout, W., "Explaining and justifying expert consulting programs," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 815-822, Vancouver, B. C., August 1981.

Swartout, W., "Gist English generator," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 404-409, AAAI, August 1982.

Swartout, W., and R. Balzer, "On the inevitable intertwining of specification and implementation," *Communications of the ACM* 25, (7), July 1982, 438-440.

Wilczynski, D., "Knowledge acquisition in the Consul system," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 135-140, Vancouver, B. C., August 1981.

Wile, D. S., *Program Developments: Formal Explanations of Implementations*, USC/Information Sciences Institute, RR-82-99, August 1982.